1-1-1988

Importance and implementation of explanation in expert systems

Kurt R. Heilbronn

Rochester Institute Of Technology
School of Computer Science

Importance and Implementation of Explanation in Expert Systems

by
Kurt R. Heilbronn

A Thesis, submitted to
The Faculty of the School of Computer Science in partial fulfillment of the requirements for the degree of Master of Science in Computer Science

Approved by:

_____
Professor J. Al Biles

_____
Professor Evelyn Rozanski

_____
Professor Peter Anderson

June 14, 1988

Abstract

Explanation is crucial in persuading others of the correctness of our beliefs to gain acceptance of our conclusion. Early research into expert systems focused on methods for reasoning. However it became apparent that the ultimate success lay in the ability to gain user acceptance by explaining the reasoning behind the conclusion. This study examines explanation from a social and an implementation standpoint. The social aspects of explanation provide insight into the role of naturally occurring explanations and listener expectations. Examination of research expert systems with explanation facilities and modifications of a simple Prolog expert system shell demonstrate the techniques required to simulate naturally occurring text. The modified shell produces improved explanations over the original shell, clearly indicating the desirability of natural appearance for gaining user acceptance.

Table of Contents

Preface

Explanation involves using mutually understood words, actions, and motives to arrive at a common understanding by reconciling differences. Without explanation we cannot convince others that our beliefs are correct. This common belief system and the ability to adjust our explanation to the listener we intend to convince is a crucial part of communication. During the development of expert systems it has become apparent their success requires explanation.

This study examines explanation in expert systems, the concepts, mechanics, and future trends toward natural explanation using natural language and discourse planning strategies.

Chapter 1 provides insight into the reason for explanation by examining explanation between humans, that is, naturally occuring explanation. This provides a basis for understanding why we explain, how we adjust our

explanations based on our audience, and the methods employed. The elements of explanation are related to expert systems, outlining current explanation techniques and desirable features that simulate natural explanation.

Chapter 2 examines the development history of expert system explanation facilities and notes the importance of considering the explanation facility during design. Knowledge representation, system building tools, and methods of reasoning are all parts of the design of a system and affect its ability to explain.

Chapter 3 departs from the conceptual level of explanation to examine explanation facilities in detail. MYCIN is a medical diagnosis system that was developed in the mid-1970's which is responsible for explanation techniques presently in use. MYCIN is examined in detail to give the reader insight into template fill-in explanation techniques and the problems associated with this method. Small Prolog systems are also presented in preparation for the implementation of such facilities in Chapter 4. Finally, research systems such as TEXT [McKE 85] and TEXTGEN [THOM 84] are examined as they represent the future trend toward generation of natural explanation, embodying many of the features considered

desirable in Chapter 1.

Chapter 4 provides a comparison of trace explanation and paragraph-form explanation by modifying a simple Prolog expert system shell. Using such a shell permits the reader to examine the mechanisms involved and gain insight into the difficulties involved in providing such a facility. Examples, results, and the code are provided for examination.

Chapter 5 presents conclusions about the study and thoughts on the future direction of explanation in expert systems.

The appendix contains complete examples of explanation output by the modified shell, and the code from the original and modified shell.

Chapter 1
Explanation

Explanation plays a role in our lives every day. We rarely think much about explanation; without it, however, our confidence in other's conclusions would suffer. Most of us would question a suggestion to delete a key file in a computer system without an explanation. Finding out that a backup copy of the file exists and that the current file is bad however, relieves a great deal of stress during the deletion as opposed to blindly deleting the file!

Explanation helps us understand someone's reasoning process. It is important to point out the difference between reasoning and explanation.

> Explanation refers to reasoning which is expressed in verbal form, and the term reasoning for internal mental processes. [GOGU 83]

Explanation has been studied since the time of Aristotle, who sought to develop truth through logic rather than through the social process of explanation. Explanation is a form of persuasion, where we attempt to convince our audience that we are correct.

Rhetoric is another area of study in persuasion. "Rhetorical study, in its strict sense, is concerned with the modes of persuasion. Persuasion is clearly a sort of demonstration, since we are most fully persuaded when we consider a thing to have been demonstrated." [GOGU 83]

For the purpose of this paper I have adopted the view of explanation and reasoning of Goguen:

> "...explanation is a phenomenon of the social world, which is transacted using language, while reasoning is a phenomenon of the internal, mental world, which leaves no directly observable trace. Specifically, an explanation is a unit of language which purports to show why a speaker believes some particular statement, and (in most cases) is intended to cause the hearer to accept this statement." [GOGU 83]

1.1 Why We Explain

We explain for various reasons such as reassuring

someone our conclusion is correct or confirming that she understands our point. The latter shows the interactive nature of explanation and the need to have common models for understanding.

Interaction is required for successful explanation. It has been shown that complexity in an explanation or lack of a model of the audience is likely to result in an unacceptable explanation [CARR 87]. To avoid this situation the speaker will often illicit information from the audience to adjust her model or assure herself that the audience is understanding. If the audience is not convinced, the speaker can adjust her mental model to talk at the level of the audience, choosing examples that they can readily relate to.

John Carroll discovered the importance of this interaction in researching issues in interfaces to advice-giving computer systems. Persons were interviewed after receiving advice and the lack of interaction came to light.

> Advisors rarely included explanations or checked whether the user understood the advice. Users often criticized advisors for not making the information meaningful. Advisory interactions that were judged successful by users tended to be ones in which the advisory strategy allowed shared control of the dialogue. [CARR 87]

The implications for computer systems, especially expert systems are obvious; users will not be likely to accept a system that does not interact with them at their level or explain its actions. Weiner, in researching explanation for expert systems, also recognized this issue.

> When designing any interactive system it is important to recognize that people perform concerted activity, e.g. conversation, only by constantly informing and conforming each other to what is to happen next. Therefore, if a computer system is to replace one of the participants in the activity, and the activity is still to succeed, the system must also be in accordance with the way people have of informing each other. [WEIN 80]

1.2 The User Model

One of the key capabilities we have as humans is the ability to form a model of our audience when explaining. This allows us to adjust our explanations to fit each person, thus making the job easier. For our computer programs this model is similar.

> The user model is what the system knows about the person interacting with it. [CARR 87]

One of the main concerns when trying to provide explanation from a computer system is determining the level

of explanation to provide, and how to make these adjustments.

The simplest approach is the Normative Model, which assumes all users to be some 'average' person. All users, of course, are not average. You probably recognized this fact the last time an 'average' user caused your computer program to fail by doing something not so 'average'! The weakness is lack of feedback.

The next approach might be to ask the user what skill level they possess. John Carroll notes "...People are notoriously bad at giving accurate descriptions of their own informational needs," [CARR 87] and he goes on to say "The obvious alternative is to infer the user's skill level from actions and responses."

The development and maintenance of user models in a computer system is an area or research that will benefit explanation systems since this model will permit the explanation facility to alter its output to the required level. This includes removing knowledge that the user already has. Weiner has pointed out that removing knowledge by assuming the user knows it can have its dangers, for instance, if the user forgot something [WEIN 80].

Therefore, a model that can constantly adapt is required.

Natural language facilities are also important in that they provide communication in a natural fashion bringing the user and computer to a common and familiar base for communication. Natural language input has been heavily researched. Natural language output has not shared the same attention but is just as important. The approach has been to work on the essentials, leaving the polishing for later. Tackling full blown natural language and user modeling issues is far beyond the scope of this study. The focus here is how to explain in an acceptable manner, that is, in the same fashion that you would expect a person to explain.

1.3 How We Explain

Before considering the organization of explanations so that we can attempt to replicate them, we should examine some ways of explaining. Factual explanation is the easiest to examine. We simply defend our statement by supplying facts; i.e. "I can proceed now because the traffic light has turned green."

Another method is to give alternatives, shifting focus between alternatives or eliminating the alternatives. This is basically weighing alternatives in a verbal form. For instance, "I did not go to the beach today. I could have, and would have gotten a nice tan, but I really should make some progress on my thesis, so I didn't."

There is also the approach where we explain why not. By process of elimination, other plausible answers are eliminated leaving our explanation as a likely one. Answers to hypothetical questions like "What if" or "Why wasn't answer X chosen" may be necessary to produce information to disprove or eliminate a choice.

1.4 Explanation's role in Artificial Intelligence and Expert Systems

With a background in how and why we explain to each other, the reasons for providing explanation in expert systems becomes clear. First, the credibility of an expert system's conclusions is greatly enhanced by persuading the user that the information is correct. If the users can compare their reasoning with that of the expert system, and see that it agrees with their own, it is likely the advice will be accepted.

Another area that explanation addresses is bias. Studies have shown that evaluation of expert system output was rated much lower when the user knew it was from a computer rather than an expert in the field. We cannot overcome the bias to the computer since the user is working at a terminal; however, a natural interface goes a long way towards removing the stigma. Both Carroll and Hayes-Roth recognize the importance of explanation facilities to the future success of expert systems [CARR 87, HAYE 83], realizing that the ultimate reason for creating expert systems was not just to see if it was possible, but also to provide a tool the user will find useful and acceptable.

> One striking lesson in recent computer science is that the capability to build a new system technology is only one condition for the potential real success of the technology. Mere technological feasibility must be augmented by empirical study of whether and how people will find new technology useful and tractable. [CARR 87]

> Although the reliability of the reasoning processes of an expert system is crucial to its ultimate success, knowledge engineers now routinely accept the fact that a variety of other parameters influence whether a system is accepted by the intended users. The nature of the discourse between the expert system and the user is particularly important. [HAYE 83]

Taking the KEE [INTE 86] system as an example, one finds that it offers wonderful graphical displays of the relationships of rules, and it is very easy to show graphs and gauges. The weakness, despite the efforts to present a

very nice looking package, is the difficulty in truly understanding the reasoning path the system chose. The user sees the rules being selected, but if he does not understand the rules and the operation of the system, then he will not understand the reasoning and will not be satisfied that it is correct.

A final reason for explanation is the ability to train users. With an explanation facility, users can compare their reasoning against that of the expert system to find where their reasoning may be incorrect.

1.5 Current Forms of Explanation

The first facilities that could loosely be deemed explanation were debugging trace packages included with the first expert systems in the late 1970s. Hayes-Roth noted that trace and debugging facilities were useful because "....such facilities help a system developer examine, follow, and understand performance of a running system" [HAYE 83].

The important thing to note is that these facilities are intended for the system developer and not the end user. The process involved in reasoning in expert systems is often

performed in a manner radically different from human reasoning and such traces can confuse rather than comfort the user.

One common extension over simple tracing facilities is the ability to reconstruct a chain of reasoning after it has been completed, rather than merely listing steps during execution [HAYE 83]. This is an improvement since it is possible to eliminate some information such as input/output processes and focus on the final solution without showing failed attempts in the process. Again, this method relies on the user being able to understand the rules being shown. Also, there is a discrepancy in the reasoning processes of humans and chaining of expert systems since humans can take educated guesses at solutions.

One level higher is the use of canned messages or templates to make the explanation appear more natural. This approach is much better than showing raw rules since it presents a readable form that the user can understand more readily. This approach is not without danger, though, since the template must fit all cases. We must ask "What are the behavioral consequences when the user inadvertently foils the template and gets garbage advice from the system?" [CARR 87].

The final and as yet unfulfilled approach is that of natural language output. Little work seems to have been done since the early 80's in this area, perhaps due to the complexity of the problem.

Since we do not fully understand the reasoning behind human explanation we have not yet been able to capture the ability to explain without limit. The work that has been done has had limited success using restricted capability for producing natural language [WEIN 80, THOM 84, McKe 85, BOYE 85, JACO 87]. Studies into the patterns of speech have shown promise. Analysis was done on transcripts of subjects thinking through problems verbally. This verbal reasoning is called a protocol. Goguen felt protocols had serious shortcomings in analyzing the aspects of explanation.

> First of all, the subject of the experiment is not socially and interactionally responsible to an audience during his speaking, as is the case in ordinary discourse. As a result, protocols can be very difficult to interpret. In contrast, if a person is trying to convince another person actually present of something, his task is to provide enough information to allow that other person to follow the argument as a whole. Furthermore, it is not clear that the structure of the subject's reasoning is adequately reflected in the protocol. [GOGU 83]

Research into naturally occurring explanation resulted in the recognition of linguistic structures used in

explanation. These structures are referred to as discourse units. "...This is a unit of spoken language, larger than the sentence, with an internal structure which can be described at least as precisely as the syntactic structure of sentences. This work in discourse analysis demonstrates that precise and describable order can be found in human speech." [GOGU 83]

This approach to explanation deals with structure and tries to cover all domains of knowledge, but the approach is restricted by its lack of an effective natural language generator. Current research in discourse presentation planning strategies and natural language generation are attempting to address the problems; however, it is not likely that a system will ever be created that can explain every domain just as humans are not capable of explaining everything. An unbreakable link exists between knowledge in the problem domain and explanation concerning that domain.

1.6 Desirable Features of an Explanation Facility

Although we have yet to produce elegant systems for explanation, a great number of features have been identified that such a system should embody. The system should be

efficient. Efficiency is a common concept in computer systems; however, the goal is user satisfaction by receiving a response to a query in a reasonable amount of time. Ease of usage is a major factor in user acceptance. If an explanation facility is difficult to use, the user will quickly abandon use of the expert system regardless of the accuracy of the information. Ease of use implies queries and responses in natural language. No further proof of this concept is required other than to observe how quickly a user will become frustrated with a complex data base query language. In addition to natural appearance the text must be familiar to the user, both in terminology and in the order that the human counterpart would present it. Given these needs, two features the user should be offered are access to the text to insure the terms are maintained and the ability to modify the appearance of the output, for instance custom templates. Not all packages offer this sort of function; as an alternative, the package should be easily incorporated and maintainable by the designer or knowledge engineer. The bottom line is a facility that can be tailored to meet user expectations.

Meeting user expectations is a difficult task. Consideration has to be given to what level of detail to provide and the type of questions the user will demand

answers to. The level of presentation falls under the umbrella of user modeling. Humans adapt their explanations to the intended audience and so should an expert system. A manager will demand a summary where an engineer may desire extreme detail. The system should ideally analyze the types of questions being asked and dynamically adapt the explanation to the percieved level desired. At a minimum the user should be able to select the level desired. Just as each user will have different needs for detail they will also have different questions. 'Why' and 'How' questions are expected; however, the system should be capable of answering questions about its operation, as well as counterfactual and hypothetical questions.

Not all features are available in any one package, and some are yet to be developed; however, they serve as useful criteria in judging the usefulness and acceptability of many packages.

Chapter 2
Explanation in Expert Systems

Explanation provides credibility for the user much as we provide credibility for our conclusions. Explanation facilities offer assistance in understanding the purpose of a question, for instance:

4) Please walk upstream along the creek looking for the outfall from which the liquid is draining. If you find one, please enter the outfall number.

WHY [i.e. WHY is it important to determine the outfall from which the liquid is draining?]

[1.01] This will aid in determining the spill basin.
It has already been established that
[1.1] The location where the spill was first reported is along-creek
Therefore,if
[1.2] the outfall from which the liquid is draining is known
then
it is definite (1.0) that the outfall from which the liquid is draining is the spill basin.
[RULE001]

Figure 2.0-1 Explanation in Response to Asking 'WHY' [HAYE 83]

The facility may be capable of providing insight to the systems operation as well, such as "Why is that a good choice of methods to accomplish that goal ?" [HAYE 83]

Facilities can permit the user to ask a variety of questions. For instance:

What is your immediate goal ?
What is the justification for that ?
Why wasn't choice B considered.
Why (was this conclusion reached)?
How did you reach this conclusion (knowledge about the system operation)?
What if X was changed to Y? (What if types of questions permit hypothetical reasoning about a problem.)
Show the alternatives. (compare two or more lines of reasoning and perhaps eliminate the others by showing weaknesses in those lines.)

This is only a partial list of the types of questions that might be answered. The task is not trivial when one consider what is involved to simply answer the question "why". Does this mean "Why did you come to this conclusion?" or does it mean something like "Why was such a slow control structure chosen?" or "Why can't the system 'backup' ?" [HAYE 83]

The full range of questions and facilities are not usually available. This may be due to the cost involved

or technical inability to provide the feature.

Explanation also acts as an aid for the system designer in detecting errors in knowledge or in the reasoning process. System development time is greatly reduced when the development tool or expert system permits explanation [HAYE 83].

The most common approach is a trace of the rules used. In rule-based expert systems this usually is a reconstruction of the rules chosen that satisfied the goal. On the simplest level the trace can be a listing of every rule and action (such as I/O to the user) the system took. This is crude since rules can often be cryptic (Figure 2.0-1); however, they can expose errors in reasoning or knowledge to aid in system debugging.

```
(P CO-ORDINATE-6
  (GOAL ^STATUS ACTIVE ^WANT CO-ORDINATE)
- (NOTIFICATION) -> (MAKE NOTIFICATION)).
```

Figure 2.0-1 Ops5 Rule
[HAYE 83]

While the some languages produce cryptic looking answers other languages such as Rosie may be quite

acceptable since the code is English-like and the meaning is clear (Figure 2.0-2).

> If the spill is a dangerous oil
> from (a source that does appear in the
> inventory), assert the location of the spill is
> known and go notify (the field team) about
> (the location of the spill).

Figure 2.0-2 Rosie Rule
[HAYE 83]

Even at the crude level of tracing, the benefits are quickly realized, that is, development time is reduced and users gain confidence in the system.

2.1 Explanation in Expert System Development

Considering the cost and time involvement to build expert systems, greater consideration must be given to explanation facilities. Hayes-Roth points out the importance of explanation facilities.

> The user interface is crucial to the ultimate acceptance of the system. [HAYE 83]

In a recent conversation, the vice-president of Level 5 Research [PER 87] indicated that the company considered explanation facilities esoteric. This seems to reflect an attitude that the cost cannot be justified. If in fact authors such as Hayes-Roth, Waterman, and Weiner are correct in stating that explanation is an indispensable feature for user acceptance, then expert systems cannot afford to be without it. Consider the cost of a very efficient and accurate expert system that is not used for lack of "user-friendliness".

Explanation must be considered from the conceptual stages right through to the final expert system. Areas to consider are knowledge representation, knowledge acquisition, types of questions that will be supported, the relationship of explanation to the reasoning method, and metaknowledge.

2.1.1 Knowledge Representation

The knowledge base contains such structures as rules or frames used to make inferences in attempting to solve a problem. These structures are typically listed if a trace feature is used.

In developing an expert system the inference engine may force us to use a particular structure that may or may not be appropriate for explanation. This is shown in Figure 2.0-1, which indicates the rule may not reflect the way the human expert normally expects knowledge to be stated. The ideal system would represent knowledge in the expected form. Since this is not the case, the ability to associate text with rules or have a separate representation for explanation is desirable. This suggests the following actions during system development:

1) Identify the intended users of the final system.
2) Make system I/O appear natural to the user.
3) Use terms and methods that the experts use.
4) Look for intermediate-level abstractions. [HAYE 83]

The last item is primarily an issue of the reasoning process; however, explanation should be addressed similarly if the facility is to be capable of intermediate or high-level summarization. Metaknowledge, that is, knowledge about knowledge, allows the system to reason about its performance by separating problem-solving knowledge (how the system works) from domain specific knowledge (knowledge the expert imparts). For instance, metaknowledge can be used to determine if a change is needed to a different inference technique if deadlock occurs in the primary inference technique or to "choose

rules entered by experts over a novice" [HAYE 83].

If properly applied, metaknowledge can be used to address questions probing the system operation such as:

> Who entered the rule ?
> Why is this rule supposed to work ?
> How well has this rule performed in practice ?
> Why are you doing an exhaustive search ? [HAYE 83]

Metaknowledge could also be used to examine user input to determine the skill level and adjust explanation to match the perceived level of ability.

2.1.2 System Building Aids

System building aids consist of programs that help acquire and represent the domain experts knowledge and programs that help design the expert system under construction [WATE 86].

Programs such as TEIRESIAS interact with the user to acquire and modify knowledge. Since the user interacts with these aids, explanation on many levels is desirable for assistance during the process. Unfortunately none have facilities that can explain the intent of the

questions it asks the user while creating rules, let alone explain why it is taking a particular approach to rule creation (metaknowledge).

Hopefully as our experience with knowledge acquisition expands, attention to explanation in these tools will be addressed.

All expert system building tools serve to speed development. Explanation speeds development by allowing the developer to see the course of reasoning and confirm that it is correct by uncovering inconsistencies in the knowledge base produced by such things as entry errors or misconceptions. Having built-in explanation tools permits quick explanation capability for the prototype system(s) so that the experts and end users can provide useful feedback during development. Since the explanation should be in terms the user is familiar with, the knowledge engineer can use this time to become familiar with the expert's terminology and create a glossary of terms to be used for explanation and other I/O.

2.2 Types of Reasoning and Explanation

Current expert systems deal with three approaches to

reasoning: retrospective, counterfactual, and hypothetical.

Retrospective is most often used, and looks in retrospect at the path chosen to solve the problem. This type of reasoning is the easiest to implement since the explanation is done by presenting the chain of rules to the user. It does not require any special knowledge in the knowledge base or special inference techniques. Polished systems eliminate user I/O and system I/O from the explanation where it is considered irrelevant and provide templates or associated text with rules to give the explanation a natural appearance.

Counterfactual reasoning is "where the system explains why an expected conclusion was not reached" [WATE 86]. This generally requires explicitly storing information in the knowledge base to disprove the expected conclusion, or use of an inference engine and explanation system capable of noting the point where the system abandoned the goal that was expected and explaining why it was abandoned.

Hypothetical reasoning, "where the system explains what would have happened differently if a particular fact or rule had been different" [WATE 86], requires the system to be able to reconstruct the steps taken to this conclusion,

determine what effect the change would have, and explain the differences or modify the knowledge base temporarily and reason again, perhaps comparing the previous conclusion to the current one or simply explaining the new conclusion, leaving the user to do the comparison.

It should be clear now that explanation and explanation facilities are a vital part of expert systems serving the user and designer and as such should be afforded the attention needed to be successful.

Chapter 3
MYCIN's Contribution to Explanation Capability

MYCIN's original goal was to produce a viable rule-based expert system; explanation was not the primary goal. It became apparent to the researchers that the ultimate acceptance of the system by users required explanation. The creation of MYCIN's explanation capability broke ground for future explanation implementations. To a great extent these methods are still used today.

3.1 The State of Explanation Systems

Upon examining expert systems and expert system building tools one will not find a wide range of capability. Current research will change this in the future, but for the most part explanation implementations follow the mold that early research systems like MYCIN created. This is primarily due to the need to produce a

commercially viable product, hence well known and less costly techniques are in use. Explanation basically falls into two categories. First, there are systems that provide textual explanation to such questions as "how" and "why". For the most part, the output is produced by filling in variables in templates. Further examination of the reasoning is provided by traversal of trees and structures left behind during a consultation. Figure 3.1-1 shows a template filled in with information derived from the consultation. For instance, the user supplied 'blood' as the site.

**WHY

IF: the site of this blood culture is blood. and
the identity of organism-1 is not known with certainty.
and
aerobic bottle growth by organism-1 has been inferred or demonstrated, and
anaerobic bottle growth by organism-1 has been inferred, or demonstrated

THEN: do the following:,
conclude that the aerobicity of organism-1 is facul (modifier: the certainty tally for the premise times .9).

conclude that the aerobicity of organism-1 is anaerobic (modifier: the certainty tally for the premise times .2)

Figure 3.1-1
Template Fill-in [SHOR 74]

Graphical displays of a hierarchical tree is another approach to explanation. The motivation follows the 'one picture is worth a thousand words' concept. If the system prints meaningful English-like text the display may be of value. Examination of Figure 3.1-2 showing output from Intellicorp's Knowledge Engineering Environment, or KEE for short, shows information that may serve to confuse the user. This is due to the fact that expert systems often select rules in an order that may contradict the order a human expert expects. If this is the case the user may feel the system is wrong in its conclusion. Further examination of the example shows that the rules are visible in the background window. If a user is not familiar with reading rules this will be a further source of confusion.

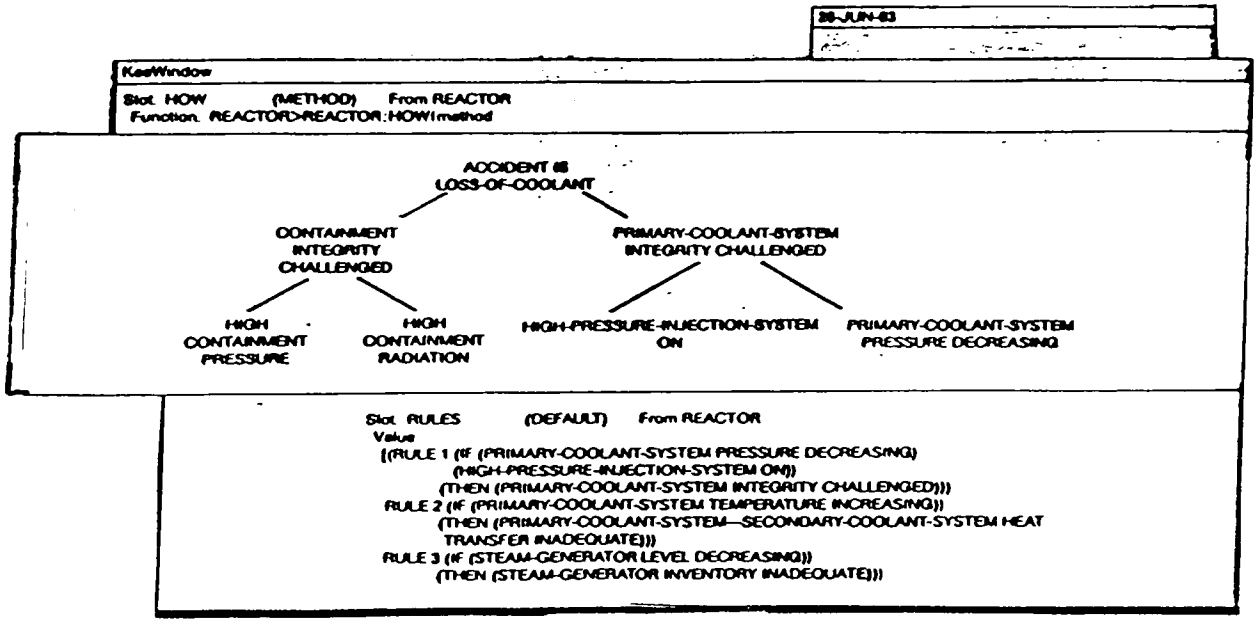


Figure 3.1-2 KEE Expert System reasoning
Intellicorp KEE users Manual

In recent years the emphasis of research shifted to natural explanations, those that appear identical to human explanation. This includes user modeling for tailoring the level of explanation to the user and providing levels of abstraction. XPLAIN (Figure 3.1-3) provides abstractions of a system's method of reasoning and general concepts in the problem domain by including meta-level knowledge about these concepts. This is illustrated in Figure 3.1-3 when the user asks for an overview. In this case the user has requested information the method used for assessing highly specific findings in the previous question. This is metaknowledge concerning the approach of the system to a portion of the problem.

Is the patient showing signs of paroxysmal atrial tachycardia with block? (yes or no):
why?

The system is assessing the highly specific findings of digitalis toxicity. Increased digitalis may cause paroxysmal atrial tachycardia with block which is a highly specific finding of digitalis toxicity.

Is the patient showing signs of paroxysmal atrial tachycardia with block? (yes or no):
overview

The system repeats the question, but the user wants an overview. This is produced by describing the method for assessing highly specific findings which was mentioned in the previous explanation.

To assess the highly specific findings of digitalis toxicity:
(1) The system assesses paroxysmal atrial tachycardia with block.
(2) It assesses double tachycardia.
(3) It assesses av-dissociation.
(4) It combines the assessments of paroxysmal atrial tachycardia with block, double tachycardia and av-dissociation.
This produces the assessment of the highly specific findings of digitalis toxicity, which is used when the system combines the assessments of the highly specific findings of digitalis toxicity, the moderately specific findings of digitalis toxicity and the non-specific findings of digitalis toxicity.

Figure 3.1-3

Justifications from a program created with XPLAIN [SWAR 83]

Further research into natural language generation focused on strategies for planning the presentation of discourse. J.L. Weiner [WEIN 80] developed a system named BLAH, which segregates knowledge into system and user knowledge so that it can reason with system knowledge and apply user knowledge to tailor the explanation (user model) by eliminating information the user presumably knows and by simulating human explanation methods such as shift in focus. Focus is required to insure the reader associates a statement with the correct reason. In Figure 3.1-4 the word 'uh' indicates a shift in focus, in this case, to explain the reasons why Peter makes less than 750 dollars. Without this indication the reasons could be incorrectly associated as an explanation of why Harry supports Peter.

(9) Well, Peter makes less than 750 dollars, and Peter is under 19, and Harry supports Peter so Peter is a dependent of Harry's. Uh Peter makes less than 750 dollars because Peter does not work and Peter is a dependent of Harry's because Harry provides more than one half of Peter's support.

Figure 3.1-4
Example of BLAH explanation [WEIN 80]

The approaches to discourse generation are examined in greater detail in section 3.7 after examining approaches to explanation in rule-based expert systems such as MYCIN in sections 3.3 through 3.5 and Prolog expert systems in section 3.6.

3.2 Short History of Development of MYCIN Explanation

MYCIN is the culmination of research at Stanford University into backward chaining rule-based expert systems in the early 1970's. The primary goal was to provide a useful system capable of diagnosing infections and prescribing a treatment. One of the sub-goals of the project was to provide the ability to "..explain its reasoning and defend its advice.."[SHOR 74]. This sub-goal eventually became a major area of research and provided the model for many expert systems to follow for explanation capability.

The evolution of the explanation facility spanned the three major uses of explanation, that of a debugging tool, a justifier, and a tutor. The RULE command was created by Shortliffe to display the rule being considered to aid in debugging. The command printed LISP rules like the following [SHOR 74]:

```
PREMISE: ($AND (SAME CNTXT GRAM GRAMNEG)
               (SAME CNTXT MORPH ROD)
               (SAME CNTXT AIR AEROBIC))
ACTION:  (CONCLUDE CNTXT CLASS ENTEROBACTERIACEAE TALLY
         .8)
```

Rules in their native representation can cause confusion. The team recognized this and "acknowledged

that if the rules were displayed in English, rather than in LISP, they would provide a partial justification of the question for the user and thereby be useful to a physician obtaining a consultation" [SHOR 74]. To provide a polished appearance of the explanation without the ability to generate natural language, each rule in the system had a required TRANS property associated with it, which served as a template. When the RULE command was invoked, a routine would fill in the blanks in the template for output. For example, given a rule of the following format:

```
PREMISE: ($AND (GRID (VAL CNTXT PORTAL) PATH-FLORA)
         (SAME CNTXT GRIDVAL (QUOTE (GRAM MORPH
          AIR))))
ACTION:  (CONCLIST CNTXT IDENT GRIDVAL .8)
```

and TRANS properties of

```
GRID:       (THE (2) ASSOCIATED WITH (1) IS KNOWN)
VAL:        (((2 1)))
PORTAL:     (THE PORTAL OF ENTRY OF *)
PATH-FLORA: (LIST OF LIKELY PATHOGENS)
```

The translation was performed by obtaining the actual values associated with the rule and placing them in the GRID in the corresponding position. Examining the rule above you will see variables such as VAL, and PATH-FLORA. Each of these corresponds to the identifiers in the TRANS property list and the instantiated values are substituted

into the text associated with GRID, in this case, "THE (2) ASSOCIATED WITH (1) IS KNOWN." The (1) and (2) in this statement corresponds to the position of the item to be inserted in the GRID portion of the rule. A sample translation might be "The list of likely pathogens associated with the portal of entry of the organism is known" [SHOR 74]. All the rules are translated in approximately the same way. Some additional complexity arises when negation is required or a certainty factor is associated with the rule.

This was the extent of MYCIN's explanation capability when the 1973 paper was prepared [SHOR 74]. The team focused their attention on explanation over the next two years based on the intuition that user acceptance required such facilities. Their intuition proved correct when, approximately ten years later, a formal study of physicians' attitudes indicated the explanation capability to be the most important facility in gaining acceptance of the system. The system was modified over the two years to create a history tree, which permitted the user to examine the reasoning chain by repeatedly asking 'why'. A 'how' function was also implemented to permit examination of alternative branches of the history tree. In addition to the explanation capability, the user interface was

modified to recognize keywords, which provided a familiar interface and improved the range of questions the user could pose.

3.3 MYCIN's Structure

In order to understand how MYCIN's explanation capabilities work (Section 3.4) a small amount of knowledge of the system structure is needed, primarily the structure of rules, properties associated with rules, and the data base that keeps track of the consultation. Each of these entities may contain information that contributes to the explanation.

The consultation program uses knowledge from the system in the form of rules and data entered from a physician. Also maintained is a dynamic data base to provide an on-going record of the consultation. This dynamic data base is a key element in the provision of explanation.

The system is goal-directed and rule-based. It has rules of the form

IF: e is known to be true
THEN: conclude that h is true with probability X

This form is consistent with other rule based systems. The rules are actually written in LISP, but a translation of a rule to English serves to make the rules more understandable. An example of a rule is:

RULE037

IF: 1) The identity of the organism is not know with certainty, and
2) The stain of the organism is gramneg, and
3) The morphology of the organism is rod, and
4) The aerobicity of the organism is aerobic
THEN: There is strong suggestive evidence (.8) that the class of the organism is enterobacteriaceae

Rules are further organized into logical groupings called <u>contexts</u>. In 1975 MYCIN had ten different context types which are summarized below:[SHOR 74]

CURCULS A current culture from which organisms were isolated
CURDRUGS An antimicrobial agent currently being administered to a patient
CURORGS An organism isolated from a current culture
OPDRGS An antimicrobial agent administer to the patient during a recent operative procedure
OPERS An operative procedure the patient has undergone
PERSON The patient
POSTTHER A therapy being considered for recommendation
PRIORCULS A culture obtained in the past
PRIORDRGS An antimicrobial agent administered to the patient in the past
PRIORORGS An organism isolated from a prior culture

Each context-type except PERSON can be instantiated more than once per consultation. Those instantiated are arranged hierarchically in a data structure termed the context tree (See figure 3.3-1).

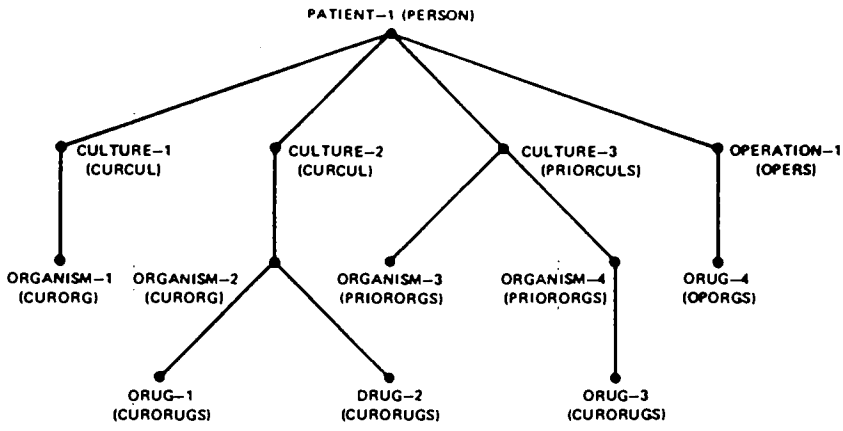


Figure 3.3-1 Context Tree [SHOR 74]

Each node in the context tree is a parameter. These parameters are capable of being retrieved and updated, and they typically contain an attribute-object-value triple. There are three possible parameter types. Single-valued parameters can take on one of several mutually exclusive values. Multi-valued parameters can take on several values. The final type is yes-no which is a two-valued Boolean parameter. The type of parameter will have an effect on the translation during explanation. For instance, the general form of translation of a parameter

and its value is THE <attribute> OF <object> IS <value> [SHOR74]. This would produce a redundant statement such as "THE FEBRILE OF PATIENT-1 IS YES"[SHOR74]. This is also the case in the original version of the expert system shell discussed in chapter 4, which presented statements such as "you finished the core is true was derived from finished core_rule". MYCIN solved this by recognizing the type of parameter and omitting the latter part to produce "PATIENT-1 is FEBRILE"[SHOR74].

Certainty factors are maintained by MYCIN. It is not relevant here how this is done, rather what is done with certainty during explanation. Based on the certainty factor, the belief is expressed by varying part of the explanation. For instance, if the certainty factor is greater than or equal to .8 on a scale of -1 to 1 then the phrase "There is strongly suggestive evidence that" is placed in front of the explanation for that parameter.

The dynamic data base maintained during a consultation allows explanation of why a question was asked as well as keeping track of the method used to obtain values of parameters. MYCIN numbers questions posed to the user so it is possible for them to ask about a preceding question any time during a consultation. The data base records

allow the system to determine which portion of a rule initiated the query so the intent of that question is known. The method used to obtain a value is useful for 'how' questions where it may be desirable to indicate to the user that he supplied the information or that the information about a culture was from a prior lab test.

The dynamic data base, context tree, and parameters provide the necessary information to handle a variety of questions when an explanation is requested.

3.4 Methods for Generating Explanations

This section examines the operation of explanation in MYCIN. MYCIN is typical of rule-based expert systems in general, so the operational principles are applicable to many existing systems whose proprietary interests discouraged us from examining their mechanisms. The goals for the explanation system and overall organization of MYCIN are instructive for those attempting to implement explanation.

The work on MYCIN suggested three major goals for the explanation capability. First, to be able to handle all relevant questions such as[SHOR 74]:

- how it made a certain decision
- how it used a piece of information
- what decision it made about some subproblem
- why it did not use a certain piece of information
- why it failed to make a certain decision
- why it required a certain piece of information
- why it did not require a certain piece of information
- how it will find out a certain piece of information

(while the consultation is in progress)

- what the system is currently doing (while the consultation is in progress)

Second, it is important to enable the user to get an explanation that answers the question completely and comprehensively. And, finally, it is necessary to make the explanation capability easy to use.

It was suggested that the explanation capability can be broken into two functions: the reasonability status checker (RSC) which is used during the consultation and the general question answerer (GQA) which is applied during and after a consultation. The GQA deals with the questions listed at the beginning of this section. It uses static domain knowledge as well as the dynamic structures discussed in Section 3.3. Meta-level knowledge about the operation of the rule interpreter is required to answer questions such as why a rule failed or why one rule is chosen over another. The operation of the RSC and GQA are examined fully in Sections 3.4.2 and 3.4.3.

After understanding that explanation is not independent of the expert system's knowledge and inference structure, it becomes obvious that explanation must be considered during the design of that expert system, especially the types of questions that are expected. The organization of the system's knowledge can make explanation very easy or extremely difficult. If the knowledge (rules) are English like in nature, only a function capable of printing the rule is needed whereas LISP rules need translation. There is also the issue of where to store descriptions of the individual rules. The designer may choose to store descriptions of the rule with the rule itself, or elect to store information about rules in general if the rules are composed of a small number of elements. The first requires entering and maintaining the description with each rule, whereas the second option only requires the initial entry for each element class. It is also a good idea to have a source of knowledge about the operation of the interpreter. This can be stored somewhere, or built into the explanation routine(s) for these types of questions. The idea of "specialists," each capable of giving a single type of explanation, has been suggested [SHOR 74]. Specialists for dealing with inferences are also useful for answering questions requiring inference and logic such as why the consultation did not use a piece of information.

3.4.1 MYCIN'S Approach to Explanation Capabilities

The previous section provided an overview of MYCIN's explanation capability. Now the reasoning status checker, general question answerer, specialists, and knowledge structure are examined further in an effort to show how MYCIN accomplishes explanation.

The explanation system consists of a number of specialists. Each specialist is designed to give a particular type of answer. The specialists are grouped by purpose.

> These specialists are grouped into three sets: one for explaining what the system is doing at a give time, one for answering questions about the system's static knowledge base, and one for answering questions about the dynamic knowledge base. The first set forms MYCIN's reasoning status checker; the second and third together make up the system's general question answerer[SHOR 74].

For a specialist to function it requires information about the consultation as well as the structure of the knowledge and/or inference technique. In order to answer questions about a specific consultation a record must be kept. This record is built during the consultation and is organized into a tree structure called the history tree (Figure 3.4.1-1). Each node in the tree represents a

goal and contains information about how the system tried to accomplish this goal, such as asking the user or by trying rules, whether or not the rule succeeded, and if not, why it failed.

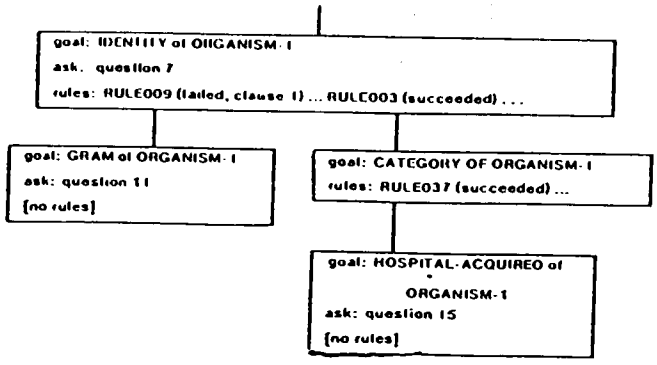


Figure 3.4.1-1 History Tree [SHOR 84]

Knowledge about rule structure is possible within each specialist by understanding that the language for MYCIN consists of a finite number of conceptual primitives such as $AND, SAME, and CONCLUDE. Having a small number of rule components also facilitates examination of rules to see which might be applicable to the explanation at hand. The knowledge of rules is derived by 'reading' them and using knowledge about the individual components to explain the rule. To explain the actions of the rule interpreter MYCIN relies on a number of specialists that know how the control structure works and what knowledge is used.

The 'specialist' responsible for filling in templates uses system maintained lists to retrieve rules that have the desired parameter in the premise portion of the rule. A template is then applied that fits the action portion of the rule to insure that a match for the desired value exists. This method employs knowledge about the control structure of the system, so the specialist need only check the action section of the rule since this is where the conclusion resides.

3.4.2 Reasoning Status Checker

The RSC (Figure 3.4.2-1) allows asking why the "piece of information being considered by the system is important" [SHOR 74]. When the system asks a question of the user, it is attempting to satisfy the current goal. By examining the history tree the ancestor goal can be determined along with the rule that invoked the current goal. To answer a 'why' question then, the system presents the goals and the rules that link them. To answer multiple 'why' questions involves following the links up the history tree to higher levels each time 'why' is entered.

```
15) Is the patient's illness with ORGANISM-1 a hospital-acquired infection?
** WHY
[i.e. WHY is it important to determine whether or not the infection with
     ORGANISM-1 was acquired while the patient was hospitalized?]
[1.0] This will aid in determining the category of ORGANISM-1
     It has already been established that
          [1.1] the gram stain of ORGANISM-1 is gramneg, and
          [1.2] the morphology of ORGANISM-1 is rod, and
          [1.3] the aerobicity of ORGANISM-1 is facultative
     Therefore, if
          [1.4] the infection with ORGANISM-1 was not acquired while the
               patient was hospitalized
     then
          there is strongly suggestive evidence (.8) that the category of
               ORGANISM-1 is enterobacteriaceae
     ALSO: there is weakly suggestive evidence (.1) that the identity of
               ORGANISM-1 is pseudomonas-aeruginosa
     [RULE037]
** WHY
[i.e. WHY is it important to determine the category of ORGANISM-1?]
[2.0]. . . in order to determine the identity of ORGANISM-1.
     It has already been established that
          [2.1] this blood culture was taken from a sterile source
     Therefore, if
          [2.2] this current organism and at least one of the list of members
               associated with the category of the organism agree with
               respect to the following properties: air conformation
     then
          There is strongly suggestive evidence (.9) that each of them is the
               identity of ORGANISM-1
     [RULE003]
** HOW 1.3
[i.e. HOW was it established that the aerobicity of ORGANISM-1 is
     facultative?]
The following were used:
[3.1] RULE027 indicated there is strongly suggestive evidence (.8) that the
     aerobicity of ORGANISM-1 is facultative
[3.2] RULE027 indicated there is weakly suggestive evidence (.2) that the
     aerobicity of ORGANISM-1 is anaerobic
Since this gave a cumulative CF of (.8) for facultative, and (.2) for
     anaerobic, it has been established that the aerobicity of ORGANISM-1
     is facultative
** HOW 1.1
[i.e. HOW was it established that the gram stain of ORGANISM-1 is gramneg?]
You said so [question 11].
```

Figure 3.4.2-1
MYCIN's Reasoning Status Checker
(user entries follow the double asterisks)

3.4.3 General Question Answerer

General questions can be asked about static knowledge

or a particular consultation. In either case there are natural language routines that recognize a number of phrases, making the system easier to use. Questions about the static knowledge base may deal with judgemental knowledge (e.g., rules used to conclude a certain piece of information) or they may ask about factual knowledge (e.g., entries in tables and lists). Sample questions are shown in Figure 3.4.3-1.

Is blood a sterile site?
What are the non-sterile sites?
What organisms are likely to be found in the throat?
Is bacteroides aerobic?
What methods of collecting sputum cultures do you consider?
What dosage of streptomycin do you generally recommend?
How do you decide that an organism might be streptococcus?
Why do you ask whether the patient has a fever of unknown origin?
What drugs would you consider to treat E.Coli?
How do you use the site of the culture to decide an organism's identity?

Figure 3.4.3-1
Sample questions about MYCIN's static Knowledge.

MYCIN is also capable of accepting a number of questions concerning the dynamic knowledge gathered during a particular consultation. The types and examples of each are listed in figure 3.4.3-2. The slot <cntxt> indicates some context that was discussed in the consultation; <parm>

is some clinical parameter of this context; <rule> is one of the system's decision rules.

1. What is <parm> of <cntxt>?

   TO WHAT CLASS DOES ORGANISM-1 BELONG?
   IS ORGANISM-1 CORYNEBACTERIUM-NON-DIPHTHERIAE?

2. How do you know the value of <parm> of <cntxt>?

   HOW DO YOU KNOW THAT CULTURE-1 WAS FROM A STERILE SOURCE?
   DID YOU CONSIDER THAT ORGANISM-1 MIGHT BE A BACTEROIDES?
   WHY DON'T YOU THINK THAT THE SITE OF CULTURE-1 IS URINE?
   WHY DID YOU RULE OUT STREPTOCOCCUS AS A POSSIBILITY FOR ORGANISM-1?

3. How did you use <parm> of <cntxt>?

   DID YOU CONSIDER THE FACT THAT PATIENT-1 IS A COMPROMISED HOST?
   HOW DID YOU USE THE AEROBICITY OF ORGANISM-1?

4. Why didn't you find out about <parm> of <cntxt>?

   DID YOU FIND OUT ABOUT THE CBC ASSOCIATED WITH CULTURE-1?
   WHY DIDN'T YOU NEED TO KNOW WHETHER ORGANISM-1 IS A CONTAMINANT?

5. What did <rule> tell you about <cntxt>?

   HOW WAS RULE 178 HELPFUL WHEN YOU WERE CONSIDERING ORGANISM-1?
   DID RULE 116 TELL YOU ANYTHING ABOUT INFECTION-1?
   WHY DIDN'T YOU USE RULE 189 FOR ORGANISM-2?

Figure 3.4.3-2

Types of Questions and examples [SHOR 74]

MYCIN must examine the user input and classify the question into one of the five types above. It then determines what specifics (clinincal parameters, rules, contexts and so forth) are being provided.

The specifics provided are used in retrieving the appropriate rules by searching lists to see which rules are applicable. The user interface comprises the first three steps in providing an answer by looking for keywords. First, breaking the text into a list of words, then determining the type of question asked, and finally determining which parameters and data structures are needed to answer the question. The portions of greatest interest to us concern classification of the question and determination of relevant pieces of knowledge. Classification identifies the specialist that will answer the question. This is done by setting flags when a pattern is matched. The classification phase also recognizes questions requiring rule retrieval. In the presence of a question requiring rule retrieval, the components of the question are broken into premise and action parts to facilitate the operation. The relevant pieces of knowledge are determined by position in the question through use of a template and are used by the specialist in construction of the explanation.

3.4.4 Answering the Question

Corresponding to each question type, there are a number of possible answer templates. The specialist for answering

questions of a given type has to check the history tree or the system's knowledge base in order to determine which of the answer templates is appropriate for a particular question. Some blanks in the answer template are filled in by the same items that are used to fill blanks in the question template. The remainder are filled by the specialist with the information that answers the question.

3.4.5 Procedure for question answering

Questions about static knowledge generally "consists of simply displaying this information after looking it up in a table or a rule [SHOR 84]." Questions about judgemental knowledge require retrieval of the relevant rules. The rule classifier splits the question into premise and action portions to compare them against lists indicating which rules reference the needed parameter. If no values were specified, the job is done and the relevant rules can be displayed without further analysis; otherwise, it is necessary to read each of the rules in the list and to eliminate those that do not mention the correct values for the parameter.

Figures 3.4.5-1 and 3.4.5-2 show factual knowledge answers and judgemental (rule-retrieval) answers.

```
**IS BLOOD A STERILE SITE?
Yes. Blood is one of those sites that are normally sterile.

** WHAT ARE THE NONSTERILE SITES?
Those sites that are normally nonsterile are: CERVIX,
        CUTANEOUS-ULCER, LOCHIA, NOSE, SKIN, STOOL, THROAT, URETHRA,
        VAGINA

** WHAT ORGANISMS ARE LIKELY TO BE FOUND IN THE THROAT?
The likely pathogens associated with throat are: STREPTOCOCCUS-
        PNEUMONIAE, STREPTOCOCCUS-GROUP-A, NEISSERIA-MENINGITIDIS.

** IS BACTEROIDES AEROBIC?
No, the aerobicity of bacteroides is anaerobic.

** WHAT DOSAGE OF STREPTOMYCIN DO YOU GENERALLY RECOMMEND?
The suggested dosage and route of streptomycin is 7.5 mg/kg
        q12h IM (or IV).

** HOW DO YOU TREAT MENINGOCOCCAL BACTEREMIA?
For treatment of NEISSERIA-MENINGITIDIS in PRIMARY-BACTEREMIA
Select therapy from among the following drugs (ranked according
        to relative efficacy, assuming uniform sensitivity):
        PENICILLIN          (.91)
        AMPICILLIN          (.9)
        SULFISOXAZOLE       (.8)
        CHLORAMPHENICOL     (.7)
        ERYTHROMYCIN        (.5)
        TETRACYCLINE        (.4)
```

Figure 3.4.5-1 Questions about Factual Knowledge

```
** WHAT DOES GU MANIPULATION TELL YOU ABOUT THE IDENTITY OF AN
   ORGANISM?
The rules listed below use:
        whether the patient has had a genito-urinary manipulative
            procedure
to conclude about:
        the identity of the organism
t56, t63, t90
Which do you wish to see?
        ** 156

RULE156

IF:  1) The site of the culture is blood, and
     2) The gram stain of the organism is gramneg, and
     3) The morphology of the organism is rod, and
     4) The portal of entry of the organism is urine, and
     5) The patient has not had a genito-urinary manipulative
        procedure, and
     6) Cystitis is not a problem for which the patient has
        been treated
THEN: There is suggestive evidence (.6) that the identity of
        the organism is e.coli
```

Figure 3.4.5-2 Rule-retrieval questions.

One property of the knowledge structure that has not been mentioned so far is the rule model. Rule models are abstract descriptions of subsets of rules. This is as close as MYCIN comes to causal knowledge. The rule models permit some abstraction in the explanation; however, it is not truly abstract due to ties to the rules (i.e., no explanation of general strategies is possible).

3.4.6 Consultation-Specific Questions

Figure 3.4.3-2 listed five types of questions that could be asked about a specific consultation. The first type of question is the simplest to answer since it only requires obtaining the value. For instance Figure 3.4.6-1 shows enterobacteriaceae as the class of organism-1.

** TO WHAT CLASS DOES ORGANISM-1 BELONG?
the category of ORGNANISM-1 is enterobacteriaceae (.9)

Figure 3.4.6-1 Questions about a parameter's value.
[SHOR 84]

The remaining types of questions require the specialists to know how the system acquires information, how it is used when acquired, and what causes rules to be tried, succeed, or fail. This is knowledge of the control structure.

Questions regarding how the system knows a value for a parameter can be determined from the history tree. Recall that the history tree records the source of the information inferred and/or supplied by the user. Figure 3.4.6-2 gives examples of answers to this type of question.

** HOW DO YOU KNOW THAT CULTURE-1 WAS FROM A STERILE SITE?
I used rule189 to conclude that this blood culture was taken from a sterile source. This gave a cumulative CF of (1.0). The last question asked before the conclusion was made was 7.

** DID YOU CONSIDER BACTEROIDES AS A POSSIBILITY FOR ORGANISM-1?
Yes. I used rule095 to conclude that the identity of ORGANISM-1 is bacteroides. This gave a cumulative CF of (.7). The last question asked before the conclusion was made was 20.

**HOW DO YOU KNOW THAT ORGANISM-1 IS E.COLI?
The following rules made conclusions about whether the identity of ORGANISM-1 is e.coli

Rule	Cumulative certainty Yes	No	Last question asked before conclusion was made
RULE021	(.47)		20
RULE084	(.55)		22
RULE003	(.74)		24

In answer to question 9 you said that the identity of ORGANISM-1 is e.coli (.3)

Figure 3.4.6-2 [SHOR 74]

There is the possibility the user could ask why a conclusion was not made. To answer this sort of question, the system determines how the conclusion could have been made and then uses knowledge of the control structure to

determine what prevented the conclusion from being reached. The user is given the reason for not reaching the conclusion, as is shown in the following example:

** WHY DON'T YOU THINK THAT THE MORPHOLOGY OF ORGANISM-1 IS COCCUS?

It is definite that the morphology of ORGANISM-1 is rod. Knowing this with certainty rules out all other values for the morphology of ORGANISM-1, including coccus.

The third type of consultation-specific question asks how a parameter was used. This specialist needs to know how a parameter can cause a rule to fail or be prevented from being considered in addition to knowing how to retrieve the rules using the parameter. Again, the history tree provides information on the relevant rules, those using the parameter in the question, and why other rules failed or succeeded because of it. Figure 3.4.6-3 illustrates this type of question and the answer.

** HOW DID YOU USE THE AEROBICITY OF ORGANISM-1?
The aerobicity of ORGANISM-1 was used in the following rules:
RULE003, RULE005, RULE084
The aerobicity of ORGANISM-1 caused the following rules to fail:
RULE035, RULE051, RULE053, RULE111.
The aerobicity of ORGANISM-1 also would have been used in :
RULE037, RULE050, RULE058, RULE086, RULE110, RULE184, RULE203, RULE204, RULE205. However none of these succeeded in the context of ORGANISM-1. If you would like an explanation for why any of these rules failed, please enter their

numbers:
** 37

RULE037 was tried in the context of ORGANISM-1, but it failed due to clause 1 ["the category of the organism is not known"].

Figure 3.4.6-3 Questions regarding how information was used. [SHOR 84]

The fourth type of question asks why the system did not find out about a particular parameter. The specialist must know what circumstances require finding out about a particular piece of information. The history tree is examined to determine why these circumstances did not arise. Figure 3.4.6-4 shows this type of question and answer. It also points out that MYCIN may not provide sufficient information to satisfy the user. It is necessary with MYCIN to ask a specific question about the area you desire further information on.

** DID YOU FIND OUT ABOUT THE CBC ASSOCIATED WITH CULTURE-1?
During the preceding consultation, I never had reason to try to find out whether the results of the CBC from the time period of this blood culture are available. It would have been necessary to know this information in order to use any of the following: RULE109. However, none of these succeeded in the context of this blood culture. If you would like an explanation for why any of these rules failed, please enter their numbers:
** 109
RULE109 would be relevant to this blood culture when applied to: ORGANISM-1.
I never had reason to try to apply RULE109 to ORGANISM-1. I would have needed to use RULE109 in order to find out whether ORGANISM-1 is a contaminant. However, during the preceding consultation, I never had reason to try to find out whether ORGANISM-1 is a contaminant.

Figure 3.4.6-4 Questions about why a parameter was not traced. [SHOR 84]

The final type of question asks about the application of a rule to a context. Three possibilities exist: "the rule told us something about the context; the rule failed when applied to that context; or the rule was never tried in that context"[SHOR 84]. Since the specialist uses the history tree, the parameter is associated with a particular context. If the user chooses the wrong context, the specialist answers the question using the correct context. Several examples are shown in Figure 3.4.6-5.

```
** WHAT DID RULE 295 TELL YOU ABOUT ORGANISM-1?
I used RULE295 to conclude that the identity of ORGANISM-1 is
        hemophilus-influenzae. This gave a cumulative CF of (.25).
        The last question asked before the conclusion was made
        was 36.

** WHY DIDN'T YOU USE RULE 112 TO FIND OUT ABOUT ORGANISM-1?
RULE112 was not executed because it would have caused circular
        reasoning when applied to ORGANISM-1. Would you like to
        see the chain of rules and parameters which makes up this
        circle?
        ** YES
I wanted to know about the identity of ORGANISM-1 because I
        try to find out the identity of the organism for all
        current organisms of the patient.
To find out about the identity of ORGANISM-1, I tried to use
        RULE021. Before I could use RULE021, I needed to know about
        a prior organism with possibly the same identity as
        ORGANISM-1.
To find out about a prior organism with possibly the same
        identity as ORGANISM-1, I tried to use RULE005. Before I
        could use RULE005, I needed to know about the aerobicity of
        ORGANISM-1.
To find out about the aerobicity of ORGANISM-1, I tried to use
        RULE031. Before I could use RULE031, I needed to know about
        the category of ORGANISM-1.
To find out about the category of ORGANISM-1, I tried to use
        RULE112. Before I could use RULE112, I needed to know about
        the identity of ORGANISM-1.
But this is the unknown parameter I sought originally.
```

Figure 3.4.6-5 Questions regarding the application of rules [SHOR 84]

3.5 Weaknesses in the MYCIN approach

One reason associated with MYCIN's inability to adequately answer some questions was the lack of "support knowledge, the underlying mechanistic or associational links that explain why the action operation of a rule follows logically from its premise. This limitation is particularly severe in a teaching setting where it is incorrect to assume that the system user will already know most rules in the system and merely needs to be reminded of their content" [SHOR 84]. This weakness was later addressed by extended rules for MYCIN by Bill Clancey [SHOR 74].

Another area of weakness deals with the ability to understand the intent of the user's question. This is an issue of natural language understanding, which was not the focus of the project.

Also absent is any sort of method for customizing the explanation for the user. This deals with the issues of user modeling and adapting the explanation's granularity to match the needs and expectations of the user. This problem was addressed by research into use of causal knowledge by Jerold Wallis [SHOR 74].

3.6 Prolog Approaches to Explanation

Prolog lends itself well to the implementation of backward-chaining, or goal-directed, inference techniques, due to the nature of its depth-first search techniques. In examining several small expert systems [SCHI 87], [SCHL 85], [BRAT 86], [BILES 87] written in Prolog, a common method of providing explanations for 'how' and 'why' questions is evident. Primarily, 'why' questions are answered with an explanation of the current rule under consideration, usually with one or more ancestor rules, in an attempt to indicate to the user why this question is needed to satisfy the goal. 'How' questions are answered using a list of values and rules that are carried as an argument and typically deal with how the conclusion was derived.

Both types of questions are answered using a list of rules that were found to apply up to the point of the question. When a rule is found to apply, it is appended to the list. This has the advantage of letting Prolog's backtracking deal with removing erroneous attempts at applying rules.

Examining these small expert systems will clarify the methods used to produce explanations. This look is intended only to gel the concepts rather than examine the gory details of implementation, which are addressed in Chapter 4.

The following systems are examined in roughly the order of explanation capability and also validity as an expert system. Note that none of these systems begins to address the complex issues of natural language generation, user modeling, or separation of the rule format and knowledge base structure from the explanation. They basically are template-type explanation systems, where the variables are filled in with information that is available at the time of the question.

Tax Advisor [SCHL 85] is capable of 'how' and 'why' questions. The validity of the program as an expert system is questionable since the knowledge is hard-coded. To a great extent the explanation capability of the program is hard-coded as well, using templates with minimal filling of information. The author justifies this method by stating that the tax rules "are justifications for themselves" [SCHL 85]. The code excerpt in figure 3.6-1 shows explanation templates used for various sections of

tax rules followed by a portion of code showing the usage.

```
/* Definition of rules to use in how and why

get_rule(1,'An individual is deemed to own stock owned
        by his family.  Section 318(a)(1).').
get_rule(2,'A spouse is a family member.  Section 318
     (a)(1)(a)(i).').
get_rule(3,'A child, grandchild or parent is a family
     member.  Section 318(a)(1)(a)(ii).')'


spouse_own(Client,Corp,N,W,H) :-
        get_rule(1,Z),
        append(W,[Z],A), .....
```

Figure 3.6-1 Templates and Sample Invocation [SCHL 85]

Answering 'how' questions involves the construction of a "list of descriptions of the Prolog rules that were successful when the program finally reached its conclusion" [SCHL 85]. It should be noted that the system will not permit the user to ask how the conclusion was reached until the final conclusion is made. Each time a rule is successful, the rule adds a description of itself as the first element in the list for generating explanation. For example, in Figure 3.6-1, the predicate spouse_own adds the description found by get_rule(2,Z) to the head of the list once it has determined both that Dean (the Client) has a spouse and the number of shares of stock the spouse actually owns. In particular, get_rule(2,Z) simply sets the first element of the list to the description:

A spouse is a family member. Section

318(a)(1)(A)(i). [SCHL 85]

The following is an example of what users might see if they asked how the system arrived at a conclusion.

Start of explanation
The number of shares of Corporation_X stock constructively owned by Dean as a result of Section 318(a)(1) is 1,000, determined as follows:
An individual is deemed to own stock owned by his family. Section 318(a)(1).
FACT: Dean is an individual.
A spouse is a family member. Section 318(a)(1)(A)(i).
FACT: Mary is the spouse of Dean.
FACT: Mary actually owns 1,000 shares.
FACT: Dean has no more family members.

END OF EXPLANATION

Figure 3.6-2 How explanation [SCHL 85]

'Why' questions are handled in a similar fashion using an additional argument in each rule. As each rule is used by the program, it is added to a list. When the user asks 'why', the head of the list is presented to explain that the program is trying to satisfy this rule. If 'why' is repeatedly asked, the system continues to show ancestor rules. If the user exhausts the list of rules used the system simply answer "This is what the Internal Revenue Code Provides! "[SCHL 85]. This system is not elegant and may not even be considered a true expert system; however,

it does present a simple view of the basics for explanation.

The expert system shell presented in Advanced TURBO PROLOG [SCHI 87] is extremely simple as well; however, it exhibits properties of a true expert system shell, that is, knowledge is separate from the inference engine and may be entered by the user at the beginning of a session.

This program is limited to answering an input of 'why' when a question is asked of the user during a consultation. A provision for 'how' questions should be simple to implement. The operation is similar to that of the Tax Advisor program in that each attribute that is found to be true (supplied by the user answering yes to a question) is appended to a list. The current object under consideration is kept as well so that it may be used in the explanation. Figure 3.6-3 is a sample consultation where 'why' was a response to a query.

```
it/has/does it round n
it/has/does it no_thorns y
it/has/does it grow_on_vines y
it/has/does it purple n
it/has/does it large y
it/has/does it green why
I think it may be
Watermelon because it has:
large
grow_on_vines
no_thorns
```

it/has/does it green y
it is watermelon

Figure 3.6-3 Example of 'why' answer [SCHI 87]

Granted, the output is crude, but a little work can produce templates tailored to the application. A complete listing of the program can be found in Appendix 3.

Finally the expert system shell from Prolog Programming for Artificial Intelligence [BRAT 86] provided the basis for the B-Shell [BILES 87], which provides an expert system shell consisting of an inference engine, Prolog expressions that permit rules to be expressed in IF THEN like structure (Figure 3.6-4) and facts (Figure 3.6-5). The knowledge is separate from the inference engine and static knowledge contained in facts can be applied without input from the user. This is an improvement over the Turbo Prolog expert system that had to ask the user for all information. The B-shell also contains explanation facilities capable of answering 'why' and 'how' questions. The explanation is not totally independent of the rule structure, however, and this can lead to awkward looking rules in order to produce explanations that are 'pretty'. An added feature of the B-shell is the ability to associate a 'silent' attribute with a rule. This attribute permits suppression of explanation for that goal.

```
mammal_rule : if
                Animal has hair
          or
          Animal gives milk

                then
          Animal isa mammal.
```

Figure 3.4-4 Example of Rule Structure [BILES 87]

```
fact : X isa animal :-  { prolog if }
     member(X, [zebra, duck, tiger, penguin ]).
```

Figure 3.6-5 Example of a fact [BILES 87]

This program works in a similar fashion to the others in this section by maintaining two structures. The first structure is a list of ancestor goals and rules, starting at the original goal and leading to the current goal or rule under consideration. It is of the form:

```
[ (peter isa carnivore) by rule3, (peter isa tiger) by
 rule5 ]
```

Example of trace list Figure 3.6-6 [BRAT 86]

The other list contains a solution tree. In this case it is an AND/OR tree [BRAT 86], and this structure is used to answer 'how' questions, where the trace structure in figure 3.6-6 is used for 'why' questions. The answer to the 'how' question is provided by traversing the tree and

formatting it into an easily readable form.

The B-shell shell is examined in greater detail in Chapter 4 where the B-shell provides the foundation for implementation of expanded explanation capability. Complete listings of the B-shell both before and after modification are provided for further examination by the reader in Appendix 1, 1a, and 2.

3.7 Discourse Strategies

Discourse is defined by Webster's dictionary as "communication of thought by words; talk; conversation." Research into discourse strategy aims to generate text in a a manner similar to a human counterpart.

It should be apparent that stylized printing of rules or templates does not reflect the methods used by humans to communicate. Explanation is tailored by humans to suit the audience. A speaker may choose to show alternatives for contrast or site examples to clarify an idea. Humans also work with visual clues and models of the listener's knowledge. If someone talks about attaching a cable to a modem and then picks up the modem, the listener can determine what a modem is (or the speaker knows the

listener and that he knows what a modem is so no further explanation is required). Research in the 80's has focused on the subject of discourse strategies and text generation. Some have studied transcripts of naturally occurring text [McKe 85], [WEIN 80], while others have addressed user modeling [HASL84]. The results have yielded various representations and methods for text generation built around semantic and object-oriented networks and hierarchies. Work by Jacobs [JACO 87] has attempted to shift the generation of text from hard-coded to the knowledge-based approaches in order to create an efficient and adaptable text generator. These approaches are examined in section 3.7.1.

3.7.1 Semantic Networks and Object-Oriented Generation

Work by Thompson [THOM 84], Boyer [BOYE 85], and McKeown [McKE 85] used a dictionary and a strategy to organize text. The lexical functions were typically done in a separate step after the required information was retrieved and organized into a structure, be it a network or a tree.

Thompson's TEXTGEN system was based on studies of "encyclopedia-like expository text.." [THOM 84]. Thompson

examined three encyclopedias to see how each presented subjects. He identified some general presentation strategies. These included:

a) Define x
b) Tell me about x
c) describe kinds of x
d) describe process p
e) analogy of x to y
f) compare and contrast x and y
g) illustrate by example x

[THOM 84]

The system is designed around frames to permit inheritance from higher levels. A slot may be expanded further (see inventor slot in Figure 3.7.1-1) if it proves useful, or it can contain canned text if further breakdown is uninteresting.

(an INVENTION-EVENT with
time = "in about 1839"
inventor = (a PERSON with
        name = "Kirkpatrick MacMillan"
        occupation = blacksmith)
class-of-invention = modification
name-of-invention = "velocipede"
modified-part = foot_pedals
description-of-invention =
        "Foot pedals enable the rider to....."
interesting-stories = ...
significance = ...)

Example of Frame for invention of bicycle Fig. 3.7.1-1
[THOM 84]

In general the process consists of four parts. First, a hierarchical structure is generated. Thompson termed these 'discourse structures' [THOM 84]. Second, daughter nodes may be generated to represent subtopics to be expanded, for instance, the inventor slot in figure 3.7.1-1. The third portion is a recursive operation. Nodes (topics) needing expansion are non-deterministically selected and expanded. The fourth part of the operation deals with keeping track of the expanded nodes and general knowledge to prevent attempts to re-expand or expand further. This 'bookkeeping' function could serve to limit output of knowledge the user already possesses if a user model existed; however Thompson did not examine this possibility. The most interesting part of Thompson's work is the inclusion of surface text generation in the object-oriented structure. The operation is similar to the four parts mentioned above except that the nodes to be expanded are noun phrases, verb phrases and the like. Thompson noted that this relieves the lexicon of a great deal of work, allowing it to focus on exceptions such as past tense. In order to achieve this, Thompson notes "The method of searching the hierarchies for methods finds exceptions before it finds general rules" [THOM 84].

McKeown also concerned herself with "what to include in

text and how to organize it" [McKE 85]. She implemented her ideas in a system named TEXT that "generates paragraph length responses to questions about data base structure." Models were developed for "three communicative goals: define, describe, and compare" [McKe 85]. McKeown's discourse strategies guide the process by determination of the next thing to say. She also realized the need for focus of attention in the text to make it coherent. J.L. Weiner [WEIN 80] also recognized this need when developing the BLAH system.

TEXT works in two stages. The first stage determines the content and structure of the discourse and is termed the 'strategic' component. The second stage, the 'tactical' component, uses a grammar and dictionary to realize in English the message produced by the strategic component.

Questions are answered using the main parts of the generator. First, relevant information is collected in a "relevant knowledge pool" then, based on the purpose and information, a discourse strategy is selected. The focusing mechanism determines the order of the answer using information in the knowledge pool. The knowledge pool serves as a device to narrow knowledge considered when

making comparisons to information that is relevant. Without this limit the explanation could include far more information than required for clear communication.

McKeown describes the basic units of discourse strategy (rhetorical predicates) as predicates used by a speaker to delineate the structural relation between propositions in a text, citing examples of "analogy", "constituency" (description of sub-parts or sub-types), and "attributive" (providing detail about an entity or event). These predicates classify propositions. Further analysis presented four patterns that were represented as schemata. They are identification, constituency, attributive, and contrastive.

The identification schema captures a strategy used for providing definitions. The constituency schema describes an entity or event in terms of its sub-parts or sub-types. The attributive schema can be used to illustrate a particular point about a concept or object. The contrastive schema is used to describe something by contrasting a major point against a negative point. The negative point is introduced first [McKE 85]. These schemata "capture patterns of textual structure that are

frequently used by a variety of people (and therefore do not reflect individual variation in style) to successfully communicate information for a particular purpose. Thus, they describe the norm for achieving given discourse goals, although they do not capture all the means for achieving these goals"[McKE 85]. The schemata act in a fashion similar to the nodes that may be expanded in Thompson's TEXTGEN system and, like TEXTGEN, may be approached recursively. The approach to selecting the proper schema is fairly straight-forward. The type of question produces a first cut into categories that contain appropriate schemata (Fig. 3.7.1.-2). For instance, a question asking the difference between airplanes and trains would map to the third category.

Requests for Definitions
Identification
Constituency

Requests for available information
Attributive
Constituency

Requests about the difference between objects
Compare and contrast

Figure 3.7.1.-2 Schemata used for TEXT question-types [McKE 85]

The final selection of schemata is based on an arbitrary depth in the hierarchical structure. This

results in constituency being chosen if little information is available about the object and identification being chosen if definitional information exists[1]. Once the schema is chosen, a function retrieves the information from the knowledge pool and formats it to an intermediate representation. This representation is eventually translated to English. Focus constraints are applied if more than one proposition in the knowledge pool applies to the schema under consideration. McKeown ordered focus constraints suggested by Sidner to maintain coherence (Figure 3.7.1-3).

1. Shift focus to item mentioned in previous proposition.
2. Maintain focus.
3. Return to topic of previous discussion.
4. Select proposition with greatest number of implicit links to previous proposition.

Figure 3.7.1-3 Order of Focus Contraints [McKe 85]

McKeown notes that the first item causes shifts as each item is introduced. On the surface it may sound hard to follow; however, this action is often found in written text. The second choice avoids cutting off the elaboration on a subject when all has not been presented. A return to a previous subject implies that the current subject is closed. The third choice is a catch-all when

---

[1]The items are ordered from most detailed to least detailed, proceeding top to bottom.

the first two choices do not apply. The final choice addresses an exception where two propositions have the same focus. The solution is to choose the proposition with the 'most mentions to previously mentioned items' [McKE 85], simulating the human tendency to group related discourse information.

When all schema are processed the tactical component translates the intermediate representations to English for presentation. The work of McKeown, Weiner, and Thompson deals with the ability to generate discourse through the use of strategies. While the approaches and capabilities differ, they share the ability to approach the problem recursively, expanding topics as needed or as applicable to the situation. The weakness of all three are their ties to the underlying structure of the knowledge base or data base. Ideally, the ability to generate text should be removed from the structure of the expert system entirely. Recent work by Jacobs [JACO 87] is aimed at this goal. It is beyond the scope of this paper to deal with these issues, so the reader is encouraged to investigate Jacob's approach.

Chapter 4
Improving Explanation in the B-shell

4.0 Introduction

A major goal of this project was the demonstration of the importance of natural appearing explanation and techniques to accomplish this. A suitable testbed for such modifications was the B-shell. This chapter discusses the original shell and how it was modified to produce paragraph-form explanations, the difficulties involved in implementing the changes, and the results.

4.1 Explanation in the Original B-Shell

The B-shell[1] handles explanation by maintaining a list of the rules, facts, and answers provided during a consultation. As each rule is examined, the succeeding portion is appended to the list. Like most Prolog

---

[1]In the interest of readability the actual code has been banished to Appendix 1, 1A, and 2. Names preceeded by the dollar sign ($) and/or containing underscores refer to procedures in the code contained in the appendix.

applications the B-shell does a depth-first search with backtracking to try alternatives that might satisfy the goal. This is particularly handy since the duty of maintaining the proper choices in the list is delegated to Prolog, which upon backtracking 'forgets' the addition to the list.

The lists involved in the explanation are contained in two variables. The first is Trace, which contains a simple list of rule names and goals used at any time during a consultation. Answer is the second variable, and it contains a tree of stylized text representing the complete actions of the program during a consultation, including items provided by the user, facts, calls to Prolog, and rules. When a user asks "why" instead of supplying an answer to a question during the consultation, Trace is displayed to give the user an idea of why the current rule is being considered. Answer is used to present the total picture after the consultation is complete and when the user wants to know how the conclusion was reached.

Presentation of the information is done in two places. The final conclusion is presented from procedures in the driver program, which invokes $_present and serves two purposes. First it presents the conclusion reached, and

then it offers the user a chance to see how the conclusion was reached. If the user elects to see how the conclusion was reached, the supporting evidence is presented using indention. The indention serves to visually group the associated reasons (figure 4.1-1). This is performed by procedure $_show.

doa thesis in os is false, which was derived by
thesis-in-field rule from
you took 709, which was computer
and
you took 706.....

Figure 4.1-1 Sample Explanation from B-shell Showing Indention

Although $_show looks a bit odd, its operation is really quite simple. The first $_show clause serves to start indention at 0 by adding the integer 0 as an argument and passing control to the remaining clauses of $_show. They in turn deal with the three possible combinations of answers, that is, "and" (Answer1 and Answer2), "but" (Answer1 but Answer2), and non-compound answers (Answer was Found). These clauses would be quite simple were it not for the silent attribute that some rules carry. If 'silent' proceeds the rule or fact, then it is not to be included in the explanation. During the consulation the silent attribute is indicated by adding the explanation to the Answer list in the form of "_ is _ was no_exp," where

the first two "_" are Prolog's don't care variable which will match anything. Testing for this condition sets up three possible conditions. Let us only consider the "and" (Answer1 and Answer2). There are four possible combinations as illustrated in the following matrix.

		Answer1: silent	Answer1: non-silent
Answer2	silent	Nothing Prints	Answer1 Prints
	non-silent	Answer2 Prints	Both Print

Figure 4.1-2 Effect of Silent Attribute on "and"

First Answer1 may or may not be silent. This is checked first. Next, Answer2 may or may not be silent. If both are silent then nothing is printed. If one or the other is silent then only the non-silent answer is printed. The final choice is that both are to be output. If this is the case then a method is required to join the two answers. This is done by presenting the first answer, then printing "and" on a separate line, then printing the second answer so that the apprearance is

Answer1_text .....

and

Answer2_text.....

The third $_show clause is identical to the "and" clauses except that it deals with explanations joined by "but" (Answer1 but Answer2). This occurs if Answer1 is found true but Answer2 was found false. The final clause deals with simple statements. If the answer passed to it is not silent then it will write the answer, followed by 'which was' and pass on the rest to $_show1. An example would be this call

$_show(doa thesis in os is false was derived by

thesis-in-field from (....)).

This results in printing

doa thesis in os is false, which was derived by
thesis-in-field rule from . . .[2]

The fourth $_show clause contributes the 'which was' section in the above text. The remaining text in this line is generated by $_show1, which is invoked in the final $_show clause. $_show1 consists of three clauses that format some introductory text around the rule (in the example above the words "rule" and "from" are added). If

---

[2]The text actually appears on one line when presented on the crt.

the answer does not match the format of a rule, then either it is some simple statement to be printed or, if uninstantiated, the word "derived" is printed. All the clauses presented so far work on text generated during the consultation and added to the list contained in Answer. The text is generated when one of the $_explore clauses matches and the type is recognized to be a rule, fact, user supplied, or computed. The format of the text is determined by the inclusion of an 'explainable' clause in the knowledge base.

The 'explainable' clause contains the format of the rule and a default template that is used if the variable is not yet instantiated. An example best illustrates this operation. The Advisor expert system included in the B-shell User Manual determines if a graduate student in RIT's Graduate Computer Science program should do a project or a thesis. One explainable phrase is "doa project in something". "doa" and "in" are operators defined as op(100, fx, [doa]) and op(90 , xfx, in). If the system is presented with input such as "doa project in What" Prolog will substitute the ubiquitous internal integer format for the uninstantiated variable "What" resulting in "doa project in _78." It should be intuitively obvious that an explanation in this form is not acceptable. To deal with this there is an explainable clause that provides a

template of the form:

explainable(doa project in _, doa project in something).

which indicates that any variable in the last word position of the expression can have the word "something" substituted. Note that the template is identical in structure to the goal, which facilitates positional substitution. From a logical viewpoint one can think of the process as comparing the goal to the template word for word. If the word in the goal is an uninstantiated variable, then the instantiated counterpart from the template is substituted. If the word in the goal is an operator or is already an atom, then nothing is changed. The actual operation requires looking at the goal and template in their internal representations. Given the operator precedence of "doa" and "in" listed above, the internal representation of each is:

Goal: doa(in(project,_78))

Template: doa(in(project,something)

Application of the univ operator (=..) in a recursive fashion yields the following lists:

Goal	Template
[doa,project in _78]	[doa,project in something]

taking the tail of the list in recursion:

[in,project,_78] [in,project,something]

An additional procedure ($_formatall) then takes each item in the goal list and feeds it back to $_format. The boundary condition occurs when empty lists are examined, at which time we start to back out. The first comparision will be the _78. This satisfies a clause checking for a variable and returns the corresponding element from the template resulting in the list [something]. The remaining items in the example are atomic; therefore, no additional substitutions are done. The returned first list looks like [in,project,something], and when univ is applied, results in the phrase "project in something." Further backing out results in the doa operator being included in the list [doa,project in something]. Again, application of univ returns 'doa project in something' which is displayed to the user.

These are the components of the current explanation facility. They are obviously straightforward and the limited number of clauses makes for easy examination and understanding of the process. The simplicity also results in some limitations.

4.2 Weaknesses in the B-shell

The B-shell falls somewhere between rule traces and template fill-in systems with its current explanation facilities. As it stands the system is typical of a great number of commercial systems with regard to explanation. It is certainly desirable to imitate human explanation as much as possible, and a natural appearance goes a long way towards increased acceptance. The explanation from the B-shell does not appear to be natural; that is, the influence of the rule structure and the indention does not appear to be text written by a human counterpart. A great deal of the unnatural appearance is caused by the massaging of rules to produce an explanation. This often produces odd phrases such as "no your specialty is anything, which was not disproved." Another problem caused by the interdependence of the rule and explanation formats not being visable to the user but rather to the knowledge engineer. As mentioned above the rules must be massaged a bit to make the explantion look reasonable. This impedes simple rule addition because the knowledge engineer has an additional burden of figuring out what the effect is on explantion. For instance, if a condition in a rule is not to be included in the explanation it is necessary to include the word "silent" before each portion

of the rule that is not to be explained (Figure 4.4.1-1).

```
project_in_field: if
        'you finished the core' and
        ( 'your specialty is' nothing
        xor
          no 'your speciality is' Anything
        ) and
        'you have' N hours and
        silent pro (N >=46) and
        major 'concentration is' MaField and
        silent 'core area' MiField and
        silent pro (MaField \== MiField) and
        minor 'concentration is' MiField and
        silent ( pro (Field = MaField)
                 or
                   pro (Field = MiField)
        )
        then
               doa project in Field.
```

Figure 4.4.1-1
Example of rule in B-shell with explanation elements embedded. Inclusion make meaning hazy.

The "silent" attribute also has a side-effect that may or may not be desired. If you think in terms of the depth-first tree that Prolog is building, then any explanation that has the node with the silent attribute as an ancestor will not be explained. If the intent was to limit some detail from being explained to the user because an assumption (correct or not) was being made that the user was familiar with this particular piece of information, then the result is not the one desired. Since whole branches of the tree are pruned it is not possible to "turn off" individual nodes and still allow child nodes to be

explained. This is an issue of user modeling with which the current shell cannot deal.

Negation often looks odd coming from the B-shell. Negation is either open world or closed world. "Is_no" is the operator for open world negation, and "no" is for closed world negation. In open world negation, "false" must be explicitly proven, whereas in a closed world, "false" is an assumption if it cannot be proven that something is true. An example of the odd appearance is the explanation "no your specialty is anything, which was not disproved." This is an explanation from a closed (and rather strange) world. The true intent of the statement is to say that no specialty was stated, and since it could not be proved that there is a specialty, it must be assumed true. It is difficult to explain assumptions, and better wording is badly needed. Currently the wording is the result of applying the clause $_invert where "true" is replaced by "false" or vice versa.

Another area that has not been addressed by the B-shell is meta-level knowledge. Meta-level knowledge could be useful for addressing the problem with negation. It can also be used to organize explanations. General concepts are useful for summaries or explanations to someone who is

not concerned with detail. There is an intuitive appeal to explaining intermediate 'why' questions with both the rules being considered and a general statement of the top level goal in terms that are familiar to the user. The B-shell currently does not address these areas.

4.3 Implementation

Improvement of the explanation component of the B-shell was undertaken to provide further experience and insight to the techniques of explanation; the intended result being an improved facility encompassing features that overcome the weaknesses discussed in section 4.2, not to provide a general explanation facility or tool set. Such an implementation would require major modification of the inference engine to accomodate the package, extensive research into natural language generation and complete user interface facilities for error checking. A key reason the B-shell was chosen as a testbed was simplicity; the explanation facilities are easily found and understood. A complete facility would increase complexity, making it difficult, if not impossible, to compare the original shell facilities to the improved ones.

4.3.1 Objectives

The proposed system addresses the weaknesses of the original shell by providing natural looking answers to questions during consultation and after. Asking why during a consultation when the user is prompted for information results in the statement of the top level goal that is being examined along with an explanation of recently considered rules. Ideally, the system would provide the last two rules in the inference chain along with the top level goal. Repeated queries would include ancestor rules higher up the chain until users are satisfied why the question was asked or reaches the end of the chain (or their rope).

The conclusion reached as the result of a consultation is presented as a statement of the top level goal and a finding. For instance,

> The intent was to determine if you should do a thesis.
> The answer is to do a thesis.

After the conclusion has been presented to the user, he is presented with the option of seeing how the conclusion was reached. The user is offered a choice of a summary or

a full explanation. If a summary is chosen, general concepts associated with the rules and supporting facts are presented, which may be thought of as a conceptual overview of the reasoning and a form of meta-level knowledge. If the user chooses a full explanation, he is presented with a paragraph detailing the rules used and the support for them. In following the advice of previous research [WEIN 80, McKE 84] the rules are explained immediately upon encountering them. The support for each entity is explained immediately. If an entity is justified by further entities then a nesting or sub-tree effect occurs.

Thesis in Field is supported by completion of the core course is supported by you took 706 and you took 709 and you took 720 and you took 744 and you took 781 and you took 809. This was all the support for you completed the core courses, and you have 46 hours and your specialty is ai and your major concentration is ai.

Figure 4.3.1-1 Embedded Explanation

Figure 4.3.1-1 illustrates the embedding of explanations. Here the Thesis-in-field rule was supported in part by completion of core courses. When the core course rule is explained, the justification is immediately presented for this sub-goal, that is, that you took courses 706 through 809. Without a phrase to indicate the completion of the sub-goal explanation, it

would be impossible to determine where the explanation for the sub-goal and parent goal separate. To accomplish this the system prints "This was all the support for completing core courses", where "This was all the support for" is the focus shift phrase and the remaining text describes the rule the preceeding text supported.

Regardless of whether a summary or full explanation is chosen, a form of user-modeling is implemented to offer varying levels of detail. The level is chosen by the user who selects a value from 1 through 10 where 1 is the least detailed and 10 the most. Each portion of the explanation has an associated level that is compared with the entered value, which acts as a threshold. If the level is less than or equal to the threshold then the text is presented. This permits the system to omit specific portions of the explanation, for instance something the user already knows, concentrating on presenting the essential information. Related to level is the 'silence' attribute. The original shell provides a method for omitting text from presentation by adding the word 'silent' before a portion of a rule, fact, Prolog call, or user supplied answer, resulting in omission of this entity and all supporting information for that entity. This implementation preserves this ability while removing the requirement of placing the silent tag in

the rule, resulting in easier to read rules.

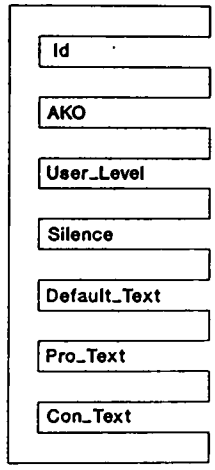
While error handling facilities were not paramount in importance, lack of an explanation required addressing. If a user selects a level of detail that is higher than any of the levels in the explanations or the explanations are all marked as silent, then the user would be baffled by the lack of explanation. If this should occur the user is informed that nothing can be shown at this level of detail and that he should consider modifying his choice.

While the user derives the greatest benefit, the knowledge engineer or designer is offered improvement as well. A key objective was the separation of explanation from knowledge to relieve the knowledge engineer from structuring rules so that they would produce explanations that were somewhat natural (see section 4.2). This implementation works with the original inference scheme while decoupling the explanation information from the rules increasing the understandability of those rules.

4.3.2 Data Structures

With a requirement to work with the original inference engine, the data structure had to provide the capabilities

desired yet fit into the original scheme where formatted text existed; the structure chosen was a frame. Full frame-based systems offer great power through inheritence and the ability to create slots that activate procedures when a value is changed, added, or deleted. For this application a full frame implementation was not required since the power could be realized with static slots. The slots chosen facilitate natural appearance, removal of the silence attribute from the knowledge base, user modeling, and summarization capability. Conceptually the explanation frame can be imagined as illustrated



Frame Structure

where the slots are defined as follows:

id: The entity currently being examined from the knowledge base. It is used as a key to the frame.

ako A-Kind-Of attribute. Used to associate the frame with general concept information for summarization purposes.

user_level A numeric value from 1 to 10 that represents a cutoff point in relation to the user's level of knowledge or interest.

silence y or n are legal values, y indicating yes and n indicating no. If silence is indicated the user will not see this piece of information nor see explanation from its children since they inherit the silence.[3]

default_text
A complete line of text to be used in the absence of any user supplied information.

pro_text Text template for positive answers.

con_text Text template for negative answers.

---

[3]The modified shell will silence children of this node like the original shell. Selective silencing requires use of the user_level slot.

The actual structure in the code has a functor name of explan_frame. The value in the ako slot of the explanation frame is used as a key to the concept_text structure. The concept text structure contains three pieces of information and is of the form,

concept_text(concept_id,line,text)

where concept_id is the same as the value found in the ako slot of the explanation frame, line is a numeric value used to order multiple lines of text, and text is the actual text to be displayed. Access routines assume the line numbers to be contiguous integers from 1 through the limit of the machine. Any added facilities for maintaining explanations would have to take this into account.

The explainable structure, discussed in detail in section 4.1, is used to instantiate variable information in templates if it was not provided by reasoning or by the user. This structure is retained in the modified shell and used to fill uninstantiated portions of the pro and con text.

The last structure added during the modification is the explan_structure. This structure might be considered a

trimmed down version of the frame. It is of the form

explan_structure(Text,Level,Silence,Concept_id,Truth)

where Text is the appropriate text with all needed default text substitutions in place, Level is the user level, Silence is the silence attribute, Concept_id is the ako link to the general concept, and Truth is either true or false. The Truth value was required due to the operation of the inference engine, which needs to know if the last item evaluated is true or false.

4.3.3 Operation

To achieve the goals set forth in section 4.3.1 a number of procedures (clauses) were modified in the original shell and several new procedures were added. These procedures were designed to interface with the original inference techniques with a minimum amount of change.

The driver program remains virtually unchanged with the exception of calling present_explan rather than $_present. This is due primarily to added features in present_explan that will be discussed shortly. One other addition to the driver is the creation of a top level goal statement. Questions from the Advisor knowledge base included in the B-shell Users Manual are basically of the form "doa What in Which" and "doa What" where What and Which can be instantiated if the user chooses to do so. An explainable clause exists for both types of questions, and these are "determine whether to do a thesis or project in your specialty area" and "determine whether to do a project or thesis" respectively. The top level goal is included to orient the user to the purpose of the consultation, should they ask 'why' when queried during a consultation.

The actual inference engine is contained in a procedure name $_explore. The name is retained in the improved version. Changes to this procedure required working with operator precedence. It was found through experimentation that much of the structure around the original text format had to remain for subsequent procedures to function properly. As an example, the original shell produced the following evaluable construct when a computed fact was found "Goal is true was computed". This structure was

replaced by a frame in the first modification based on the assumption that "is true was computed" was no longer relevant to the explanation due to the information contained in the frame and that the source of the information was more useful to someone debugging the system. This resulted in structures that could not be separated to produce explanation in the desired fashion. As a result of this discovery the final version of the shell uses the structure "explan_frame(...) was computed", where explan_frame(...) is the explanation frame and its contents, 'was' is an operator, and 'computed' is an atom. This evaluates properly and the extra 'was' and 'computed' are ignored during presentation. In examining the inference engine, the changes are focused on four clauses that may be thought of as primitives since they deal with specific cases and not the compound structures. They are those dealing with Prolog callable goals, facts, computed facts, rules, and user supplied answers.

Prolog callable goals cannot be handled by the frame structure due to the near infinite possibilities for native Prolog calls. To accomodate this a structure identical to the frame is created on the fly that consists of the Prolog call if the answer is true or the Prolog goal and the

phrase "evaluated to false" if false. Since native Prolog calls are typically uninteresting if not confusing, it is assumed that this detail should be suppressed except for the most curious user; therefore, it is given a user level of 10.

The remaining primitives are associated with information that is loaded in the data base. The knowledge engineer can determine this finite number of entities and create appropriate frames and concepts so that the procedure get_explan can retrieve the frame and create the appropriate explan_structure. It should be noted that the pro and con text slots in the frames contain both text and operators so that $_format_goal can manipulate the text and replace missing variables with default text. This operation was described in detail in section 4.2 and remains unchanged here.

The procedure $_no_more_exp serves the same purpose as the original shell, that is, to see if there is anything left to explain. It is modified since there are multiple constraints that determine if the particular element of the explanation is silenced. In the previous shell the atom 'silent' was placed before a portion of a rule, and when encountered during inference it would result in replacing

the answer with the tag no_exp. An example of this would be the rule for completing the core courses in the Advisor knowledge base. This rule indicates the core courses are complete if courses 706, 709, 720, 744, 781, and 809 were taken (not to mention passed!) If, in the thesis_in_field, rule the consequent 'you have finished the core' were preceeded by 'silent' then you would not see the core completed explanation nor any explanation about the individual courses since they are children of the core compeleted rule. This feature is retained in the modified shell however the inclusion of a user level complicates the decision of what is silent and what is not. In order to resolve this a decision was made to give the silence attribute priority over user level. If silence equals 'y' then that node and all its children are considered silent. If silence is 'n' then the user level is compared to the threshold entered by the user. If the threshold is numerically lower than the user level from the explanation structure then this particular phrase is omitted without affecting the children. In the second case the children must be examined to see if they are silenced as well. This checking also applies when presenting the explanation. Fortunately the procedure for showing the explanation uses $_no_more_exp which reduces complexity at that phase.

Presenting the explanation expanded from one proceedure to four: present_explan, new_line, $_show, and summarize. Present_explan is fairly simple and serves more as the boss to invoke the other routines. Present explanation prints the conclusiong with the top level goal and the answer found, for example

```
The intent was to determine whether to do a thesis in ai
The answer is to do a thesis in ai.
```

The user is then prompted to determine if showing how the conclusion was reached is desired. If the user replies 'y' (yes) then $_show is invoked. $_show runs the detailed explanations. The user is prompted to select either a summary or a detailed explanation. Once chosen the user must enter the threshold level. At this point it is entirely possible that the user selected 1 as a threshold but all frames have a user value higher than 1 which results in the user being baffled by the absence of an explanation. To avoid this $_no_more_exp is called and if no explanation is detected a warning message is printed to select a higher threshold. The message includes the previously entered threshold to assist the user in a proper choice. The other possibility is that the knowledge engineer entered 'y' in the silence attribute of all items considered in the inferencing. In this case the program

detects that the user level is higher than 1 and warns that the knowledge engineer may need to be contacted to resolve the problem.

If explanation exists to be seen then the previous entry is examined. If summarization is chosen procedure summarize is invoked which takes each element of the solution and determines if it is to be displayed, and if it is, retrieves the concept structures in line number order and prints them. Procedure new_line is used to count the number of characters printed on a line so far and generates a new line character to avoid words being broken at the end of each line. New_line is also used in the detailed explanation for the same purpose. If the detailed explanation was chosen then the procedure involved printing the text in the explanation structures if they are not silent or above the threshold. The same rules apply here as for determining if no more explanation ($_no_more_exp) exists. The detailed explanation includes linking words and phrases to further the natural appearance, making it appear to be a paragraph. If conjunctions exist they are linked by 'and' and disjunctions by 'but'. The phrase 'This was the support for ' is placed in front of the previous rule's explanation to produce phrases such as "This was the support for you took the core courses". The

inclusion of these phrases serves to indicate a shift in focus back to the previous topic so that the user is always sure of the associations.

4.4 Results

The changes to the B-shell were implemented to achieve the design goals set forth in section 4.3. Perhaps the greatest hinderance to achieving the stated goals was working within the structure of the original shell since this required fitting the structure to the problem rather than finding the best solution and insuring the shell created the proper structure to generate natural language or allow easier identification for focusing portions of explanation. Even so, the goals set forth were achieved.

Perhaps the most important goal for the changes was to separate the explanation dependent portions from the rules in the knowledge base. This was achieved by moving the silence attribute to the frame structure. The original rules in the Advisor program remain unchanged except for the removal of the word 'silent' from the rules that originally contained it. If so desired, the rules could be changed to any format the knowledge engineer desired, hopefully improving readability. The only tie to the

orignal rules is the explainable clause. This clause, while not part of the actual rule, is used to extract instantiated information to be placed in the explanation so this clause must match the element of the rule under consideration. The use of the explainable clause was prompted by the desire to use existing routines to the greatest extent possible, as a result the default text slot in the explanation frame structure was not used.

An additional goal was to include a statement of the top level goal with intermediate questions and the final conclusion. The goal statement is now included with intermediate why questions and the conclusion, for instance

Is it true: your specialty is ai? w {w indicates 'why'}
Investigating whether to do a thesis in ai

This was your question

Is the response to an intermediate question and a conclusion appears as

The intent was to determine whether to do a thesis in ai
The answer is do a thesis in ai.

The remaining goals are not easily separated for demonstration purposes. There are now two separate routes that the user can select for an explanation: full and summary.

The closest resemblance to the original B-shell is the full explanation. If you elected to see how the original B-shell reached the conclusion you would recieve an indented trace of the rules (Fig. 4.4-1). This trace did not offer any dynamic options to the user as to what would be shown and what would not. The choice of what to show was made by the knowledge engineer when the rules were written.

```
Is it true:  your specialty is ai? y.

doa thesis in ai is true

Would you like to see how? y.

doa thesis in ai, which was derived by thesis_in_field rule from
    you finished the core, which was derived by core_done rule
  and
    you have 44 hours, which was computed
  and
    your specialty is ai, which was told
  and
    major concentration is ai, which was derived by concentration rule from
        you took 782, which was computed
      and
        ai seminars totaled 4 hours, which was derived by sem_hour rule
```

Figure 4.4-1 Explanation from Original B-Shell

The modified shell retains the ability for the knowledge engineer to select particular items to be silenced when the explanation frames are created however the user can dynamically choose a level of detail to further limit what is seen. The most obvious change from the original

B-shell is the format of the text presented to the user. The objective was to make the output appear natural to a user. The paragraph format achieves this with some degree of success. Working within the confines of the B-shell's inferenece technique made it impossible to identify relationships among the pieces of the explanation so phrases to link the information together remain crude. The method for word wrapping at the end of a line is also crude so the paragraph form could be improved. Despite these weaknesses the comparison of the two explanations (Figure 4.4-1 and the example that follows below) demonstrates the higher likelyhood of acceptance by a user.

```
The intent was to determine whether to do a project in ai
The answer is do a project in ai

Would you like to see how? y.

Enter an "s" if you would like to see a summary, any other letter
to see a full explanation.    Enter choice: f.


Enter a number from 1 to 10 to select the amount of detail you
want to see, 1 being the least detailed.  Enter level of detail: 10.

do a project in ai which is supported by the core courses were completed
 which is supported by you took 706 and you took 709 and you took 720 and
you took 744 and you took 781 and you took 809 This was the support for
the core courses were completed.     and do a thesis in your field and
you have 46 hours of courses and 46>=46 and your (major concentration is ai)
 which is supported by you took 782 and you have 4 seminar hours in ai
 which is supported by sem_tot(ai,4) This was the support for
you have 4 seminar hours in ai.     This was the support for
your (major concentration is ai).     and ai\==os and your (minor concentration
 which is supported by you took 810 and you have 4 seminar hours in os
 which is supported by sem_tot(os,4) This was the support for
you have 4 seminar hours in os.     This was the support for
your (minor concentration is os).     and ai=ai This was the support for
do a project in ai.

More solutions?
```

Having the ability to dynamically select a threshold at the time of presentation presented some interesting problems. First, it is possible that the user could select a threshold that is below anything that can be seen. To deal with this error checking is done to see if nothing will be shown to the user and they are warned if this is the case with

> The explanation for this solution cannot be shown because the information is suppressed. Please try a higher value detail level than 1 since this may permit some explanation to be presented.

where the level indicated (1 in the example above) is a reminder of the value the user last entered. This is not the only case that can prohibit any explanation from being presented. Perhaps the knowledge engineer does not understand the usage of the silence slot or a particular inference chain happens to select frames that all have silence set to 'y'. If this is the case you can select the highest level of detail and still not see an answer. In this case an assumption was made that the user could not deal with this problem and the following message is issued

> All explanation is silenced. Contact the Knowledge Engineer to correct this problem

Summarization is the other route the user may elect to take. Unlike full explanations, summarization cannot

provide negative text. It was decided during design that a concept is a concept and that the user can infer why the concept fails to be satisfied by rerunning the consultation and comparing the summary against a full explanation.

The following example is a summarization from the modified shell. The user is again given the opportunity to suppress parts of the summarization by entering a threshold. Both the full summary and a partially suppressed version are presented below:

> Doing a thesis is the preferred choice if you have completed the core courses with at least 44 hours and specialized in a core area. The project option only requires the major field. This is supported by the core courses consist of courses 706, 709, 720, 744, 781, and 809. Completion of a core is required for all Masters Candidates. A minimum number of hours are required for both options, this is calculated from your transcripts. If you have a specialty you should be doing a thesis in your specialty area. Determination of a concentration is done by examining the courses to see if a core area course and the complement course for that area were completed (the number of credit hours determines if this is a major or minor). This is all the support for do a thesis in ai

4.4.1 Improvements

While the modifications met the design goals there are areas which could benefit from improvement. The concepts used in summarization and the user level slot are

forms of metaknowledge there could be a greater level of involvement, such as metaknowledge about explanation orgainization or knowledge about how the system works. To provide answers relating to system operation, the latter requires a major overhaul to the user interface so that different types of questions could be asked. Dynamic knowledge of the user is also desirable. In this version of the shell a generic user was assumed for setting user levels in the frames as well as silence. If a full frame package were employed each user could have frames defining their knowledge in the subject area and this information could be used to dynamically choose what to supress and what to show. Disregarding the limitation of the generic user the system cannot unsilence explanation. As was noted in section 4.4 the system can only issue an error message if all portions of the explanation have the silence slot set to 'y'. A desirable feature would be a routine to reset the silence attribute to 'n' if the user desired an explanation after receiving the message. Finally, the appearance of the text can be improved. The current word wrapping operation deals with the complete text in the explanation structure rather than a word at a time. If the complete text would pass the end of the current line it is placed on the next line. If the text is long this can leave a very short string on one line making the paragraph

appear ragged.

Despite some weaknesses the improved explanation facilities demonstrated techniques for providing explanation and the need for natural appearance in gaining user acceptance.

Chapter 5
Conclusions

The background, examples, and implementation of explanation facilities in expert systems has demonstrated the need for natural looking explanation if expert systems are to be accepted. Comparison of explanation from the original B-shell and the modified version should convince the reader of this. It should be recognized that the changes here were for illustrative value and not intended to be a polished toolset; however, there are some areas that would benefit from further research.

First, a maintenance facility is needed to create the explanation frames, concepts structures, and explainable structures. While the rules and explanation components were separated, the maintenance task remains complex. The current system makes no attempt to do any consistency checking to insure all pieces required for an explanation are present. Another factor to consider is setting the

explanation frames silence slot and user_level slot. In retrospect it is not clear how the knowledge engineer can make such decisions if he does not understand all the possible combinations and the implications of omitting various pieces of information at various user levels. This suggests a need to improve the user modeling capability so that the choice is made in a dynamic fashion based on a set of rules about user behavior, user knowledge levels, and the importance of each item in the knowledge base. A step in the correct direction would be the inclusion of information about each user that is permitted to use the system to tailor the silence and user level attributes. This implies that the silence and user level slots would be removed from the explanation frame, and inheritance would be implemented so that a dynamic determination could be made to show or suppress a portion of the explanation.

A final area that will need addressing is the issue of uncertainty. As this project concludes, uncertainty handling is being added to the B-shell in another project. The current facility only deals with two valued logic and cannot express its strength of belief in a portion of the explanation. To address this, a set of phrases such as those associated with certainty factors in

MYCIN [SHOR 74] could be added preceding a statement. The addition of uncertainty to the B-shell as well as the modifications for explanation may have pushed the B-shell to its useful limits as a research environment. If further explanation research is desired, it may be worthwhile to implement a frame-based system which would interface with the frame-based approach to explanation, removing limitations imposed by the original B-shell on the design of the explanation facility.

5.1 Future Directions

A thorough review of the literature of the last decade has shown a definite trend toward competent natural language generation and the ability to plan discourse so that it appears to have been generated by a human counterpart. Each component of this process, such as natural language generation, user modeling, and discourse planning by themselves constitute major areas of research. In my opinion explanation facilities that are totally domain and system independent will never be seen. The strongest argument in support of this belief can be found not in the computer science arena but by examination of ourselves. Each of us is capable of using explanation techniques, however we cannot explain everything, because

we do not know everything. For this same reason expert systems cannot reason about the world in general but must focus on a specific domain. XPLAIN [SWAR 83] exemplifies this relationship by using the automatic programmers output to build concepts for a specific domain to be used by the explanation facility. Explanation systems will ultimately consist of expert systems for each of the major functions, one system that knows how to generate natural language structures, one that knows about user modeling, one concerned with human behavior, and so on, passing information among themselves via a blackboard to simulate the approach of the brain to this complex task.

Hopefully this project has given the reader some insight into the workings of explanation from both the human standpoint and from the implementation standpoint, making clear the need for natural explanations if the systems are to gain wide acceptance and a permanent place in society. It is hoped that research will continue to actively pursue the components that will make up the explanation facilities of the future.

References

[SHOR 74]
Buchanan, B.G.; Shortliffe, E.H. "Methods for generating explanations in Rule-Based Expert Systems: the MYCIN experiments of the Stanford Heuristic Programming Project." Addison-Wesley Publishing Co., Reading, Ma. 1974 pp. 338-362

[BILES 87]
Biles, J. Al "B-Shell User's Manual"
Rochester Institute of Technology, School of Computer Science, 1 Lomb Memorial Drive, Rochester, N.Y., 1987.

[BORL 86]
Borland International, Inc. "Turbo Prolog Users Manual." Borland International, Inc. 4585 Scotts Valley Drive, Scotts Valley, Calif. 1986 218 pages.

[BOYE 85]
Boyer, M.; Lapalme, G. "Generating sentences from semantic networks" Natural language understanding and logic programming. Proceedings of the first international workshop, Rennes, France 18-20 Sept. 1984 pp. 181-189

[BRAT 86]
Bratko, Ivan. "Prolog Programming for Artificial Intelligence" Addison-Wesley Publishing Company, Inc., 1986.

[CARR 87]
Carroll, John M.; and McKendree, Jean. "Interface Design Issues for Advice-Giving Expert Systems." Communications of the ACM, Vol. 30 Number 1, Jan. 1987 pp. 14-30

[GOGU 83]
Goguen, J.A.; Linde, C.; and Weiner, J.L. "Reasoning and Natural Explanation." International Journal of Man-Machine Studies Vol. 19 , 1983. pp. 521-559

References

[HASL 84]
Hasling, D.W.; Clancey,W.J. ; Rennels,G. "Strategic explanations for a diagnostic consultation system." International Journal of Man-Machine Studies Special issue:Developments inexpert systems, Part 2. Vol. 20, Number 1, Jan 1984. pp. 3-19

[HAYE 83]
Hayes-Roth,F;Lenat,D.B.;Waterman,D.A. "Building Expert Systems" Addison-Wesley Publishing, Inc. Reading, MA. 1986. pages 546

[HUTC 84]
Hutchins,M.W.; Dusek,L. "How vocabulary is generated determines speech quality." Computer Design Vol. 23, 2 Feb 1984 pp. 73-88

[INTE 86]
Intellicorp, Inc. KEE User's Manual Intellicorp, Inc. 1986

[MCKE 85]
McKeown, K.R. "Discourse Strategies for Generating Natural Language Text." Artificial Intelligence Vol. 27, Number 1, Sept 1985 pp. 1-41

[SCHI 87]
Schildt, Herbert "Advanced Turbo Prolog Version 1.1" Borland-Osborne/McGraw-Hill Programming Series, Osborne-McGraw-Hill, Berkeley, Calif. 1987. Pages 310.

[SCHL 85]
Schlobohm, D. "Tax advisor: a Prolog program analyzing income tax issues." Dr. Dobbs Journal Vol. 10, Number 3, pp. 64-92

[SWAR 83]
Swartout, W.R. "XPLAIN: A system for creating and explaining expert consulting programs." Artificial Intelligence, Vol. 21, Number 3, Sept 1983, pp. 285-325

References

[THOM 84]
Thompson, C.W. "Object-oriented text generation."
First conference on Artificial Intelligence applications,
Denver, Co. 5-7 December 1984, pp. 524-529. IEEE Computer
Society Press, 1984

[WATE 86]
Waterman, D.A. "A Guide to Expert Systems."
Addison-Wesley Publishing Co. Inc., Reading, Ma. 1986
Pages 356.

[WEIN 80]
Weiner, J.L. "BLAH, A System Which Explains its Reasoning."
Artificial Intelligence, Vol. 15, 1980. pp. 19-48

[PERS 87]
Personal communication with the Vice President of Level-5
Research, 1978.

Extended Bibliography

The following extended bibliography is included for the reader with a desire to learn more about explanation and natural language. The readings will provide breadth and depth in these areas.

Extended Bibliography

Buchanan, B.G.; Shortliffe, E.H.
Customized explanations using causal knowledge
1984, pp. 371-387 Addison-Wesley Publishing Co., Reading, Ma.

Buchanan, B.G.; Shortliffe, E.H.
Specialized Explanations for dosage selection.
1984, pp. 363-370 Addison-Wesley Publishing Co., Reading, Ma.

Buchanan, B.G.; Shortliffe, E.H.
Extensions to rules for explanation and tutoring 1984, pp. 531-568 Addison-Wesley Publishing Co., Reading, Ma.

Software tool speeds expert systems Systems and Software
Vol. 4 Number 8 Aug 1985 pp. 71-76

Applications of Artificial Intelligence IV
Proceedings of SPIE International Society of Optical Engineers, Conference held at Innsbruck, Austria, 15-16 April 1986
Vol. 657, 15-16 Apr. 1986 pp.

Personal Consultant Plus - Users Guide
TI Part No. 2539262-0001, Revised August 1986
April 1986. Texas Instruments Inc.

Personal Consultant Plus - Reference Guide
TI Part No. 2539262-0001, Revised August 1986
April 1986. Texas Instruments Inc.

Extended Bibliography

PC Scheme - A simple, modern LISP users guide TI Part No. 2537904-0001, Revised August 1986, December 1985
Texas Instruments Inc.

Personal Consultant Easy, Reference Guide
Texas Instruments Inc.

Personal Consultant Easy, Getting Started
August 1986, Texas Instruments Inc.

Knowledge Engineering System Knowledge Base Author's Reference Manual for KES Release 2.1
Software Architecture and Engineering, Inc. February 1986

Knowledge Engineering System Knowledge Base Author's Reference Manual. Supplement for KES Version 2.2
Software Architecture and Engineering, Inc. June 1986

Adorni,G.; DiManzo, M.; Giunchiglia,F. Adaptive natural language generation.
Artificial Intelligence and Information-control Systems of Robots. Proceedings of the third international conference, Smolenice,Czechoslovakia 11-15 June,1987 pp. 119-122

Aida, H.; Tanaka, H.; Moto-oka, T.
A Prolog extension for handling negative knowledge New Generation Computing
Vol. 1 Number 1 1983 pp. 94-98

Appelt, D.E.
Planning english referring expressions.
Artificial Intelligence
Vol. 26 Number 1 April 1985 pp. 1-33

Extended Bibliography

Barry, Richard
Expert Systems in Prolog. PC AI
Vol. 1 Number 2 Summer 1987 pp. 22-29

Basile, J.D.
Artificial Intelligence: the human-computer interface ?
Journal of FORTH application and research (Institute of Applied Forth Research, P.O.Box 27686, Rochester, NY 14627)
Vol. 4 Number 2 1986 pp. 169-172

Berry, D.C.; Broadbent, D.E.
Expert sytems and the man-machine interface
Expert Systems (GB)
Vol. 3 Number 4 Oct. 1986 pp. 228-231

Berry, D.C.; Broadbent, D.E.
Expert sytems and the man-machine interface (Part 2)
Expert Systems (GB)
Vol. 4 Number 1 Oct. 1986 pp. 18-28

Biegl, C.; Foxvog, D.; Kawamura, K.
Distributed expert systems in Prolog.
Proceedings of the Eighteenth southeastern symposium on system theory, Knoxville, TN, 7-8 April 1986. IEEE Computer Society Press. 7-8 April 1986 pp. 279-284

Bundy, A.
Catalogue of Artificial Intelligence Tools
pp. 168pp. Springer-Verlag New York Inc., New York, NY. 1986.

Carbonell, J.G. et al
The XCALIBUR Project
IJCAI 83: Proceedings of the Eigth International Joint Conference on Artificial Intelligence
1983 pp. 653-656

Extended Bibliography

Carbonell, Jaime G., et al Andriole, Stephen J.
The Role of User Modeling in Natural Language Interface Design Applications in Artificial Intelligence
1985, pp. 213-226
Petrocelli Books, Inc.

Carbonell, Jaime G., et al Andriole, Stephen J.
First Steps Toward an Integrated Natural Language Interface (XCALIBUR) Applications in Artificial Intelligence
1985, pp. 227-243
Petrocelli Books, Inc.

Caviedes, J.; Bourne, J.R.; Shiavi, R.
MEKEAS: a medical knowledge engineering assitstant. (Frontiers of engineering and computing in health care 1984)
Proceedings of the sixth annual conference of the IEEE Engineering in Medicine and Biology Society, Los Angeles, Ca. 15-17 Sept. 1984, pp. 55-61

Chandrasekaran, B.
Generic tasks in knowledge-based reasoning: A level of abstraction that supports knowledge acquisition, system design, and explanation
in Methodologies for intelligent systems. Proceedings of the ACM SIGART international symposium (Knoxville, Tn. Oct 22-24, 1986), Oct 22-24, 1986 pp. 2-7

Clocksin, W.F.; and Mellish, C.S.
Programming in Prolog
1981, Springer-Verlag Berlin Heidelberg

Cockett, J.R.B.
File handling for detail and extent and for subtasks in the implementation of decison processes Information Sciences
Vol. 37 Number 1-3 Dec 1985 pp. 157-168

Extended Bibliography

Cynar, Larry; Mueller, Donald; and Paraoczai, Andrew
Computers design networks by imitating the experts.
Data Communications, April 1986,
pp. 137-145

Davis, Randall;Lenat, Douglas B.
Knowledge-based Systems in Artificial Intelligence 1982,
pp. 490. McGraw-Hill, Inc., New York, NY

Dere, M.A.; McKeown, K.R.
Using focus to generate complex and simple sentences 10th International conference on Computational Linguistics, 22nd annual meeting of the assoc. for computational linguistics. 2-6 July 1984 pp. 319-326 Proceedings of Coling 84, Standford, Ca.

DeVries, P.H.; Robbe
An overview of medical expert systems Methods of information in medicine Vol. 24 Number 2 April 1985 pp. 57-64

Erdman, H.P.
The impact of an explanation capability for a computer consultation system Methods of information in medicine (Germany)

Vol. 24 Number 4 Oct 85 pp. 181-191

Erman, L.D.; Scott, A.C.; London, P.E.
Separating and integrating control in a rule-based tool.
Proceedings of the IEEE workshop on principles of knowledge based systems, Denver, Co., 3-4 Dec 1984, pp. 37-43

Feiner, S. Wein, M.
Research issues in generating graphical explanations
in Graphics Interface '85. Proceedings of the 11th Canadian conference (Montreal, Quebec, Canada, May 27-31, 1985), pp. 117-123

Extended Bibliography

Fichtelman, M.
Level five Research: Insight 2+ AI Expert, Vol. 1,
Number 4, Dec 1986, pp. 73-78

Finin, Timothy W.;Joshi, Arvind K.; and Webber, Bonnie Lynn
Natural Language Interactions with Artificial Experts
Proceedings of the IEEE
Vol. 74 Number 7 July 1986 pp. 921-938

Floyd, Michael
Suitable for Framing. Turbo Technix
Mar-April 1988 pp. 80-88

Gaines, B.R.
From ergonomics to the fifth generation: 30 years of
human-computer interaction studies
Computer Compacts (Netherlands)
Vol. 2 Number 5-6 Nov 84 - Jan 85 pp. 158-161

Ganascia, J.-G.
Explanation and justification in expert systems. Computers
and Artificial Intelligence
Vol. 4 Number 1 1985 pp. 3-13

Ganascia, J.G
Explanation and justification in expert systems. Computers
and artificial intelligence
Vol. 4 Number 1 1985 pp. 3-13

Ginsberg, M.; Ginsberg, M.
Counterfactuals
Artificial Intelligence, Vol. 30 Number 1, Oct 1986
pp. 35-80

Extended Bibliography

Goodall, A.
Prolog - The rules to becoming an expert. New Electronics
Vol. 17 Number 19 2 oct 1984 pp. 79-82

Gross, G.
The KEE system and some application examples.
Proceedings of expert systems: available hard- and
software, Mol, Belgium, 18-19 June 1986, 17pp

Hardt, S.; Rapaport, W.
Recent and current artificial intelligence research in the
department of computer science, State Univ. of New York at
Buffalo
AI Magazine
Vol. 7 Number 2 Summer 1986 pp. 91-100

Harrington, R.J.
The expert system development tool KES
proceedings of expert systems: available hard- and
software, Mol, Belgium, 18-19 June 1986 17pp

Hayes, P.J., and Carbonell, J.G.
Multi-Strategy Parsing and its Role in Robust man-Machine
Communication. Technical Report CMU-CS-81-118.
Carnegie-Mellon University Computer Science Department
May 1981

Helman, D.H.;Bennett, J.L.
Theories of explanation: Expert systems and simulation
Texas A & M University Symposium on Human Factors in
Management Information
Systems, College Station, TX, 8-9 Oct 1986

Helman,D.; Bennett,J.; Foster,A.
Simulations and symbolic explanations.
in Methodologies for intelligent systems. Proceedings of
the ACM SIGART international symposium (Knoxville, Tn. Oct
22-24, 1986)

Extended Bibliography

Hovy, E.H.; Schank, R.C. Bara, B.G.; Guida, G.
Language generation by computer
in book: Computational models of Natural Language Processing
pp. 165-195. Amsterdam, Netherlands

Hudson, D.L.; Cohen, M.E.; Deedwania, P.C.
EMERGE: a rule-based expert system implemented on a
microcomputer Microcomputer applications
Vol. 3 Number 3 1984 pp. 79-83

Jones, K. ;Sparch Bramer, M.A.
Natural Language Interfaces for Expert Systems
Proceedings of the Fourth Technical Conference of the
British Computer Society Specialist Group on Expert Systems
18-20 Dec. 1984 pp. 85-94
Cambridge University Press, 1985

Josephson, J.R.; Chandrasekaran, B.; Smith, J.W. Jr.
Assembling the best explanation
Proceedings of the IEEE workshop on principles of
knowledge based systems, Denver, Co., 3-4 Dec 1984.
IEEE Computer Society Press. 3-4 Dec 1984
pp. 185-190

Keen, M.J.R. and Williams, G. Bramer, M.A.
Expert System Shells Come of Age Proceedings of the
Fourth Technical Conference of the British Computer
Society Specialist Group on Expert Systems
Vol. Number 18-20 Dec. 1984 pp. 13-22
Cambridge University Press, 1985

Kidd, A.L.; Cooper, M.B.
Man-machine interface issues in the construction and
use of an expert system
International journal of man-machine studies
Vol. 22 Number 1 Jan 1985 pp. 91-102

Extended Bibliography

Kipps, J.R.
An overview of ROSIE: a programming environment for expert systems Proceedings WESTEX-86, IEEE Western Conference on Knowledge-based Engineering and expert systems, Anaheim, CA. 24-26 June 1986 pp. 88-94

Kraft, A.; Winston, P.H.; Prendergast, K.A.
XCON: an expert configuration system at Digital Equipment Corporation (in book AI Business, the commercial uses of artificial intelligence), pp. 41-49
MIT Press, Cambridge, MA. 1984 324pp.

Lawrence, K.
Artificial Intelligence in the Man/Machine interface
Data Processing
Vol. 28 Number 5 June 1986 pp. 244-246

Lehmann, H.
Automatic construction of discourse
10th International conference on Computational Linguistics, 22nd annual
meeting of the assoc. for computational linguistices
2-6 July 1984 pp. Proceedings of Coling 84,
Standford, Ca.

Lehmann, W.; Bennett, W.
Human language and computers
Computers and the Humanities
Vol. 19 Number 2 Apr-Jun 1985 pp. 85-88

Marcus, C.
Prolog programming: Applications for database systems, expert systems, and natural language systems. 325pp.
Addison-Wesley Publishing Co., Reading, Ma. 1986

Extended Bibilography

McDonald, D. Cercone, N.
Dependency Directed Control: It's Implication for Natural Language Generation
Computational Linguistics
1983 pp. 111-130
Pergamon, New York, New York. 1983

McDonald, D.D.
Surface generation for a variety of applications [text]
AFIPS Conference Proceedings: 1985 National Computer Conference, Chicago IL5-18 July 1985, pp. 105-110

McKeown, K.R.
Using focus to constrain language generation
Digest of papers COMPCON Spring 85. Thirteenth IEEE Computer Society Int. Conference. Technological leverage: a competitive necessity. 25-28 Feb. 1985 pp. 261-274
IEEE Computer Society Press

McKeown, K.R.
Focus Constraints on Language Generation.
IJCAI 83: Proceedings of the Eigth International Conference on Artificial Intelligence, 1983, pp. 582-587

McKeown, K.R.
The TEXT System for Natural Language Generation: An Overview
Proceedings of the 20th Annual Meeting of the Association for Computational Linguistics, pp. 113-120

Merritt, D.
Forward chaining in Prolog
AI Expert, Vol. 1 Number 3, Nov 1986, pp. 75-78

Michie, D.
RuleMaster: a second-generation knowledge engineering facility
First conference on Artificial Intelligence applications, Denver, 5-7 December 1984, pp. 524-529
IEEE Computer Society Press, 1984

Extended Bibliography

Mizoguchi, F.
Prolog based expert system
New Generation Computing
Vol. 1 Number 1 1983 pp. 99-104

Mulders, L.
Expert systems and Prolog
Proceedings of expert systems: available hard- and software, Mol, Belgium, 18-19 June 1986, pp. 6

Naish, L.
Negation and control in Prolog [Lecture notes in computer science;No. 283], 119pp.
Springer-Verlag New York Inc., New York, NY, 1986

Nakagawa, H.J.
Prolog programming transformations and tree manipulation algorithms. Journal of logic programming.
Vol. 2 Number 2 July 1985 pp. 77-91

Narain, S.
MYCIN in a logic programing environment.
Digest of Papers COMPCON Spring 84', Twenty-eigth IEEE Computer Society International Conference, San Francisco, CA. 27 Feb - 1 Mar, 1984
pp. 192-197. IEEE Computer Society Press 1984.

Narain, S.
MYCIN: implementing the expert system in LogLisp IEEE Software
Vol. 2 Number 3 May 1985 pp. 83-88

Neches, R.; Swartout, W.R.; Moore, J.
Enhanced maintenance and explanation of expert systems through explicit models of their development
Proceedings of the IEEE workshop on principles of knowledge based systems, Denver, Co., 3-4 Dec 1984. IEEE Computer Society Press. pp. 173-183

Extended Bibliography

Rada, R.
Trends in AIM [Artificial intelligence in medicine]
SIGBIO Newsletter
Vol. 6 Number 5 June 1984 pp. 8-10

Roizen, I.; Pearl, J.
Learning link-probablities in causal trees
Report CSD-860095, University of California Computer Science Dept., Los Angeles, Ca.
June 1986, pp. 4

Rossi, G.
Uses of Prolog in implementation of expert systems New Generation Computing (Japan)
Vol. 4 Number 3 1986 pp. 321-329

Sandell, H.S.H.; Bourne, J.R.
Expert systems in medicine: a biomedical engineering perspective CRC Critical Reviews in Biomedical Engineering

Vol. 12 Number 2 1985 pp. 95-129

Sandell, H.S.H.; Bourne, J.R.; Shiavi, R.
GENIE: a generic inference engine for medical applications
CRC Critical Reviews in Biomedical Engineering Vol. 12 Number 2 1985 pp. 66-69

Schwartz, R.L.; Shostak, R.E.
Programming for AI
PC Tech Journal
Vol. 4 Number 12 Dec 1986 pp. 191-199

Shaw, I.
The application of artificial intelligence principles to teaching and training
Interactive learning international (GB)
Vol. 3 Number 4 Dec 1986 pp. 20-22

Extended Bibliography

Shmueli, O.; Tsur, S.; Zfira, H. Kerschberg, L.
Rule support in Prolog (in Expert database systems, Proceedings of the first international workshop, Kiawah Island, SC, OCT 24-27, 1984) 701pp.
Benjamin/Cummings Publishing Co., Menlo Park, Ca, 1986.

Sinha, A.; Caplan, L.; Desal, B.; Evans, M.; Hier, D.; Hill, H.
Making decision support systems self-explanatory
Proceedings COMPSAC 84. The IEEE Computer Society Eigth International Computer Software and Applications Conference, Chicago, IL, 7-9 Nov. 1984, pp. 358-362
IEEE Computer Society Press

Smith, Edward T.
A Debugger for Message-based processes
Software-Practice and Experience
Vol. 15 Number 11 November 1985 pp. 1073-1086

Sterling, L.; Shapiro, E.
The art of Prolog: advanced programming techniques. [MIT Press Series in Logic Programming]. 427pp. MIT Press, Cambridge, Ma 1986.

Swarthout, William R.
XPLAIN: A System for Creating and Explaining Expert Consulting Programs
Artificial Intelligence
Vol. 21 Number 1983 pp. 285-325

Swartout, W.R. Wojcik, A.S.
Knowledge needed for expert system explanation.
in 1985 National Computer Conference Proceedings [AFIPS conference proceedings]
Vol. 54 Number July 15-18,1985 pp. 93-98
AFIPS Press, Reston, Va. 1985.

Extended Bibliography

Swartout, W.R.
Knowledge needed for expert system explanation
AFIPS Conference Proceedings: 1985 National Computer Conference, Chicago IL5-18 July 1985. pp. 93-98

Various Authors
Digest of papers COMPCON Spring 85. Thirteenth IEEE Computer Society Int. Conference. Technological leverage: a competitive necessity. 25-28 Feb. 1985 pp. 261-274
IEEE Computer Society Press

Williams, R.S.
The desktop expert
Proceedings COMPCON Fall 84'. The Small Computer (R)Evolution, Arlington, Va. 16-20 Sept 1984. IEEE Computer Society Press. 16-20 Sept 1984 pp. 16-19

Wilson, W.G.
Prolog for applications programming
IBM Systems Journal
Vol. 25 Number 2 1986 pp. 190-206

Yager, R.R.
Explanatory models in expert systems.
International Journal of Man-machine studies Vol. 23 Number 5, Nov. 1985 pp. 539-549

Appendix 1

Original B-Shell Code

Code unique to the original B-shell. This code existed in combination with the common code in Appendix 1A to provide the functions of the original B-Shell.

```
1,$p
/*  Top-level driving procedure
        After getting a high level goal to justify or reject,
        look first for a positive answer, and if one is found
        prohibit negative answers from being found (i.e. if I
        can prove it true, I can't also prove it false).
        If no positive answer is found, then look for negative
        answer(s) to report to show why it is false.
        If no answers can be found at all, report that and quit.
*/

expert :-
        ( _ : _                         /* If a knowledge base has not  */
        ;                               /*   been loaded yet, then      */
          loadkb                        /*   ask the user for one       */
        ),
        $_getquestion( Question),       /* Get something to work on     */
        $_initialize,                   /* KB Init Prolog routine       */
        ( $_answeryes( Question)        /* Try to find yes answer       */
        ;
 $_answerno( Question)          /* O/W look for no answer       */
        ;
 nl,
          write('No more solutions could be found.'),
          nl
        ).


$_answeryes( Question) :-
        $_markstatus( negative),        /* No positive answer yet       */
        $_explore( Question, [], Answer),/* Look with trace empty       */
        $_positive( Answer),            /* Accept only positive ans     */
        $_markstatus( positive),        /* Positive answer found        */
        $_present( Answer),             /* Display the answer           */
        nl,
        write( 'More solutions? '),     /* Look for more solns?         */
        getreply( ynw, Reply),          /* Get yes or no from user      */
        Reply == no.                    /* yes backtrack to explore     */

$_answerno( Question) :-
        retract(no_positive_answer_yet), /* No positive answer yet so   */
        !,                               /*    can have negative answer */
        $_explore( Question, [], Answer),/* Look with trace empty       */
        $_negative( Answer),            /* Accept only negative ans     */
        $_present( Answer),             /* Display the answer           */
        nl,
        write('More negative solutions? '), /* Look for more solns?     */
        getreply( ynw, Reply),             /* Get yes or no from user   */
        Reply == no.                       /* yes backtrack to explore */


$_markstatus( negative) :-
        assert(no_positive_answer_yet).  /* Set bit until pos ans found */

$_markstatus( positive) :-
```

```
        retract(no_positive_answer_yet), /* Reset when pos ans found  */
        !
      ;
true.                  /* Succeed if pos answer already found  */


$_getquestion( Question) :-            /* Get question from user       */
      ( questform(_),
        nl,
        write('Ask one of the following questions:'),
        nl,
        !,
        $_write_quests
      ;
true
      ),
      nl,
      write('Question, please:'),
      nl,
      read( Question).

$_write_quests :-
      questform(Quest),
      write(Quest),
      nl,
      fail.
$_write_quests.


$_initialize :-
      init(X),
      $_run_init(X),
      fail.
$_initialize.

$_run_init([First|Rest]) :-
      !,
      call(First),
      !,
      $_run_init(Rest).
$_run_init([]) :-
      !.
$_run_init(Thing) :-
      call(Thing),
      !.


loadkb :-
      nl,
      write('What knowledge base file do you want to load? '),
      read(File),
      ( _ : _,
        !,
        write('Replace or add to the existing knowledge base? (r, a) '),
        read(Ans),
```

```
          ( Ans == a,
            consult( File)
          ;
 reconsult( File)
          )
        ;
 reconsult( File)
        ).

clearws :-
        abolish( wastold, 3),
        abolish( end_answers, 1),
        abolish( lastindex, 1),
        assert( lastindex( 0)),
        abolish( false_goal, 1),
        abolish( true_goal, 1),
        abolish( computed, 1),
/*      abolish( derived, 1),*/
        abolish( no_positive_answer_yet, 0).

dumpws :-
        listing( lastindex),
        listing( end_answers),
        listing( wastold),
        listing( computed),
/*      listing( derived),*/
        listing( false_goal),
        listing( true_goal).

traceon :-
        trace_flag,
        write( 'Chaining is already being traced.'),
        nl,
        !.
traceon :-
        assert( trace_flag),
        write( 'Chaining will be traced.'),
        nl.

traceoff :-
        trace_flag,
        abolish( trace_flag, 0),
        write( 'Chaining will not be traced.'),
        nl,
        !.
traceoff :-
        write( 'Chaining was not being traced.'),
        nl.
:
```

```
/* This version of explore is a modification and extension
        of the Bratko version from p. 331.  The modifications
        are chiefly performance enhancements to avoid some of
        the redundant recursive calls to explore and to place
        cuts where they can limit the clauses that apply to a
        given goal.

        The extensions include a number of additional operators
        to handle more interesting rule constructions and new
        hidden fact type to allow a pseudo-forward chaining.
        The new operators are:

                no Goal : Universal negation, Goal always fails
                        forall(X, ~f(X))

                is_no Goal : Existential negation, Goal fails once
                        exists(X, ~f(X))

                Goal1 xor Goal2 : Exclusive or, Or with a cut
                        (Goal1, ! ; Goal2)

                (pro Goal) : Prolog callable Goal, Meta call
                        aall(Goal), !

                computed(Goal) : Fact asserted via "forward chaining"

                silent Goal : No explanation generated for Goal

        The reason "no" and "is_no" were chosen as negation
        operators was to avoid ambiguity with Prolog's "not"
        operator.  Of the two, "no" behaves most like Prolog's
        "not" in that it only succeeds if Goal cannot succeed.
        The "is_no" operator fails only if the goal cannot fail.

        The "xor" operator complements the original incuusive or
        and behaves like an or-with-a-cut to let the user cut down
        on backtracking when the first solution in a disjunction
        is OK.

        For Prolog callable goals, the goal should be enclosed
        in parentheses to insure that the precedences of the
        built-in Prolog operators are maintained.
        The reason the "pro" operator exists, instead of just
        sending goals to Prolog automatically, is to avoid
        ambiguity with "useranswer" goals.

        A "computed" fact is presumably asserted by a prolog rule
        called via the "pro" operator.  This allows some
        forward-chaining-like behavior by putting facts into a
        working storage area "invisibly."  The format of these
        computed facts is ugly:
```

```
                computed(Fact).

        rather than the prettier user-defined forms.  This is
        because these facts are part of working storage, not
        the knowledge base, and they need not be explained.
        Newly computed facts are put into the knowledge base via
        postfact(Fact), which asserts a computed(Fact) clause
        only if a unifiable one does not already exist.
*/

:- op( 900, xfx, :).
:- op( 870, fx, if).
:- op( 880, xfx, then).
:- op( 570, xfy, or).
:- op( 570, xfy, xor).                  /* Exclusive or: Or with a cut  */
:- op( 550, xfy, and).
:- op( 540, fy, silent).                ** Silent goal: No explanation  */
:- op( 530, fy, no).                    /* Negated goal: Closed world   */
:- op( 530, fy, is_no).                 /* Negated goal: Open world     */
:- op( 520, fx, pro).                   /* Meta-hack: Send to Prolog    */
:- op( 800, xfx, was).
:- op( 300, fx, 'derived by').
:- op( 600, xfx, from).
:- op( 600, xfx, by).
:- op( 550, xfy, but).


/* explore( Goal, Trace, Answer).

        Explore a Goal to arrive at an Answer considering the Trace
        of previously answered goals.

        Can be done if Goal is a fact, the consequent of a rule,
        a conjunction of subgoals, a disjunction of alternative
        goals, a negated goal, a Prolog-callable "meta-goal"
        or a user-answereable "question" goal.
*/

$_explore( Goal1 and Goal2, Trace, Answer) :-   /* Conjunctive goal     */
        $_exp_trace( 'Enter and ', Goal1 and Goal2),
        !,
        $_explore( Goal1, Trace, Answer1),      /* Do first term        */
/*      $_assign_exp( RawAns1, Answer1),
*/
     $_continue( Answer1, Goal1 and Goal2,   /* Handle the rest      */
              Trace, Answer).

$_explore( Goal1 xor Goal2, Trace, Answer) :-   /* Exclusive or behaves */
       $_exp_trace( 'Enter xor ', Goal1 xor Goal2),
       !,                                       /*   like an or with a  */
       (                                        /*   cut to avoid bktk  */
         $_explore( Goal1, Trace, Answer1),     /* Explore first one    */
         $_positive( Answer1),                  /* If it's true, done   */
         !,
         $_assign_exp( Answer1, Answer)
```

```
        ;
 $_explore( Goal2, Trace, Answer2),    /* O/W try other side   */
          $_positive( Answer2),               /* If it's true, done   */
          !,
          $_assign_exp( Answer2, Answer)
        ;
 !,                                   /* O/W call the false   */
          $_false_or(Goal1, Goal2, Trace, Answer)/*   handler          */
        ).

$_explore( Goal1 or Goal2, Trace, Answer) :-    /* Inclusive "or" is ;  */
        $_exp_trace( 'Enter or ', Goal1 or Goal2),
        !,
        (
 __explore( Goal1, Trace, Answer1),    /* Explore first one    */
          $_positive( Answer1),               /* If it's true, done   */
          $_set_true( Goal1),
          $_assign_exp( Answer1, Answer)
        ;
 $_explore( Goal2, Trace, Answer2),    /* O/W try other side   */
          $_positive( Answer2),               /* If it's true, done   */
          $_set_true( Goal2),
          $_assign_exp( Answer2, Answer)
        ;                                    /* O/W it's false       */
          not true_goal( Goal1),             /* Be sure no positive  */
          not true_goal( Goal2),             /*   answers were found */
          !,
          $_false_or(Goal1, Goal2, Trace, Answer)
        ).

$_explore( silent Goal, Trace, Answer) :-       /* No explanation       */
        $_exp_trace( 'Enter silent ', Goal),
        !,
        $_explore( Goal, Trace, Answer1),       /* Explore the Goal and */
        $_make_silent( Answer1, Answer).        /*   cut explanation    */

$_explore( no Goal, Trace, Answer) :-           /* Closed world negate  */
        $_exp_trace( 'Enter no ', Goal),
        !,                                      /*   all(X, not X)      */
        (
 $_explore( Goal, Trace, RawAnswer1),  /* Explore the Goal     */
          $_assign_exp( RawAnswer1, Answer1),
          $_positive(Answer1),                  /* If there's any way   */
          !,                                    /*   it's true, then    */
          $_invert( no, Answer1, Answer)        /*   return a false     */
        ;
 explainable( Goal, Form),             /* O/W try to build a   */
          $_format( Goal, Form, Phrase,[],Vars),/*   nice explanation   */
          !,                                    /*   with instantiated  */
          Answer                                /*   variables          */
            (no Phrase is true was 'not disproved')
        ;
!,
          Answer =                              /* O/W build an ugly    */
            (no Goal is true was 'not disproved') /* explanation        */
```

```
        ).

$_explore( is_no Goal, Trace, Answer) :-        /* Open world negation  */
        $_exp_trace( 'Enter is_no ', Goal),
        !,                                      /*   exists(X, not X)   */
        (
 $_explore( Goal, Trace, RawAnswer1),  /* If find a negative   */
          $_assign_exp( RawAnswer1, Answer1),
          $_negative( Answer1),                 /*   answer, then there */
          $_set_false( Goal),                   /*   exists a Goal that */
          $_invert(is_no, Answer1, Answer)      /*   is false           */
        ;
 not false_goal( Goal),                 /* If no false answers  */
          !,                                    /*   can be found, then */
          $_explore( Goal, Trace, RawAnswer1),  /*   get a true answer  */
          $_assign_exp( RawAnswer1, Answer1),
          $_positive( Answer1),                 /*   and invert it      */
          $_invert(is_no, Answer1, Answer)
        ).

$_explore( pro Goal, Trace,
                Goal is Truth was called) :-    /* Prolog-provable goal */
        $_exp_trace( 'Enter pro ', Goal),
        !,
        (
 call( Goal),                           /* Send Goal to prolog  */
          !,                                    /* No backtracking cuz  */
          Truth   true                          /*   it might fail      */
        ;                                       /* If call fails return */
          Truth = false                         /*   false, don't fail  */
        ).

$_explore( Goal, Trace, Goal is true was        /* Goal is a fact       */
                'given as a fact') :-
        fact : Goal,
        $_exp_trace( 'Found fact ', Goal).

$_explore( Goal, Trace, Goal is true was        /* Goal was computed &  */
                computed) :-                    /*   asserted by prolog */
        computed(Goal),                         /*   using postfact     */
        $_exp_trace( 'Found computed ', Goal).

$_explore( RawGoal, Trace,                      /* Goal is consequent   */
                Goal is TruthValue was          /*   of a rule          */
                'derived by' Rule from Answer) :-
        Rule : if Condition then RawGoal,
        $_exp_trace( Rule, ' rule ', RawGoal),
        $_explore( Condition,                   /* Satisfy Condition    */
                [RawGoal by Rule | Trace],      /* Set up for why quest */
                Answer),
        $_format_goal( RawGoal, Goal),
        $_truth( Answer, TruthValue).

$_explore( Goal, Trace,
                Goal is Answer was told) :-     /* User-askable goal    */
```

```
        askable( Goal, _),
        $_exp_trace( 'Enter askable ', Goal),
        !,                                      /* Can only ask 1 way   */
        $_useranswer( Goal, Trace, Answer).


$_continue( Answer1, Goal1 and Goal2,           /* Rest of conjunction  */
                Trace, Answer) :-
        $_positive( Answer1),                   /* First part OK        */
        $_explore( Goal2, Trace, Answer2),      /* Explore the rest     */
        ( $_positive(Answer2),                  /* If rest is OK...     */
          Answer = Answer1 and Answer2           /* ...return conjunc    */
        ;
 $_negative(Answer2),                   /* O/W return answer    */
          ( Answer2 = A21 but A22,
            Answer - Answer1 and Answer2        /* ...return conjunc    */
          ;
 Answer = Answer1 but Answer2           /* ...return conjunc    */
          )
/*        Answer = Answer2                      /* from rest            */
        ).

$_continue( Answer1, Goal1 and Goal2,           /* If couldn't get yes  */
                _, Answer1) :-                  /*   or no return       */
        $_negative( Answer1).                   /*   a no answer        */


$_false_or(Goal1, Goal2, Trace, Answer) :-      /* Handles false "or"s  */
        $_get_neg_ans( Goal1, Trace, Answer1),
        $_get_neg_ans( Goal2, Trace, Answer2),
        Answer = Answer1 and Answer2.            /* Return both falses   */

$_get_neg_ans( Goal, Trace, Answer) :-
        $_explore( Goal, Trace, RawAnswer),     /* Explore first term   */
        $_negative(RawAnswer),                  /*   for a false answer */
        $_set_false( Goal),
        $_assign_exp(RawAnswer, Answer).

$_get_neg_ans( Goal, Trace, Answer) :-
        not false_goal( Goal),
        !,
        (
 explainable( Goal, Form),
          $_format( Goal, Form, Phrase,[],Vars),
          !,
          Answer = (Phrase is false was 'not disproved')
        ;
 Answer = (Goal is false was unprovable)
        ).


/*$_truth(Var, false) :-                        /* Just in case...      */
**      var(Var),
        Var = false,
        !.
```

```
*/

$_truth( Question is TruthValue was Found,      /* Pull out truth val   */
                        TruthValue) :- !.

$_truth( Answer1 and Answer2, TruthValue) :-
          $_truth( Answer1, true),
          $_truth( Answer2, true),      /* If both sides are true...    */
          !,
          TruthValue = true             /* ...then the whole thing is   */
        ;
 TruthValue = false.           /* O/W it's false               */

$_truth( Answer1 but Answer2, TruthValue) :-
          $_truth( Answer1, true),
          $_truth( Answer2, true),      /* If both sides are true...    */
          !,
          TruthValue = true             /* ...then the whole thing is   */
        ;
 TruthValue = false.           /* O/W it's false               */


$_positive( Answer) :-
        $_truth( Answer, true).

$_negative( Answer) :-
        $_truth( Answer, false).


$_set_true( Goal) :-
            true_goal( Goal),
            !
          ;
 assert( true_goal(Goal)).

$_set_false( Goal) :-
            false_goal( Goal),
            !
          ;
 assert( false_goal(Goal)).

$_invert( no, Goal is true was Found,   /* Invert true and false for    */
        no Goal is false was Found).    /*   negated goals              */
$_invert( no, Goal is false was Found,
        no Goal is true was Found).
$_invert( is_no, Goal is true was Found,/*IInvert true and false for    */
        is_no Goal is false was Found). /*   negated goals              */
$_invert( is_no, Goal is false was Found,
        is_no Goal is true was Found).

/*$_make_silent( Goal is TruthValue was 'derived by' Rule from Answer,
                Answer) :-
        !.
*/
$_make_silent(Goal is Truth was Found,
```

```
                Goal is Truth was no_exp).

$_assign_exp(RawAnswer, Answer) :-
        RawAnswer = (RawGoal is Truth was Found),
        $_format_goal( RawGoal, Goal),
        !,
        Answer = (Goal is Truth was Found).
$_assign_exp(Answer, Answer).


$_format_goal( RawGoal, FormGoal) :-
          explainable( RawGoal, Form),
          $_format( RawGoal, Form, FormGoal,[],Vars),
          !.
$_format_goal( Goal, Goal).

$_exp_trace( Heading, Goal) :-
          trace_flag,
          write( Heading),
          write( Goal),
          nl,
          !
        ;
 true.

$_exp_trace( Head1, Head2, Goal) :-
          trace_flag,
          write( Head1),
          write( Head2),
          write( Goal),
          nl,
          !
        ;
 true.
:
```

```
1,$p
/* Display the conclusion of a consultation and a how question
*/

$_present( Answer) :-
        nl,
        $_showconclusion( Answer),              /* Give the answer      */
        nl,
        nl,
        write( 'Would you like to see how? '),  /* Ask user for how     */
        getreply( ynw, Reply), !,
        ( Reply == yes,
          $_show( Answer),                      /* Give decision chain  */
          nl,
          !
        ;
 true                                   /* O/W just quit        */
        ).


$_showconclusion( Answer1 and Answer2) :-       /* Conjunctive answer   */
        !,
        $_showconclusion( Answer1),
        write(' and '),
        $_showconclusion( Answer2).

$_showconclusion( Answer1 but Answer2) :-       /* Conjunctive answer   */
        !,
        $_showconclusion( Answer1),
        write(' but '),
        $_showconclusion( Answer2).

$_showconclusion( Conclusion was Found) :-      /* Simple answer        */
        write( Conclusion).


$_show( Solution) :-                            /* Show driver          */
        nl,
        $_show( Solution, 0),                   /* Indent 0 to start    */
        !.

$_show( Answer1 and Answer2, Indent) :-         /* Writes out complete  */
        !,                                      /*   Answer nicely      */
        (
          Answer1 = ( _ is _ was no_exp),
          Found1 = no_exp,
          !
        ;
          $_show( Answer1, Indent)
        ),
        (
$_no_more_exp( Answer2),
          !
        ;
( Found1 == no_exp,
```

```
            !
           ;
  AndIndent is Indent - 2,
            tab( AndIndent),
            write( and),
            nl
           ),
           $_show( Answer2, Indent)
         ).

$_show( Answer1 but Answer2, Indent) :-          /* Writes out compeete  */
         !,                                     /*   Answer nicely      */
         (
  Answer1 = ( _ is _ was no_exp),
           Found1 = no_exp,
           !
         ;
  $_show( Answer1, Indent)
         ),
         (
  $_no_more_exp( Answer2),
           !
         ;
  ( Found1 == no_exp,
            !
           ;
  AndIndent is Indent - 2,
            tab( AndIndent),
            write( but),
            nl
           ),
           $_show( Answer2, Indent)
         ).

$_show( Answer was Found, Indent) :-
         ( Found == no_exp,
           !
         ;
  tab( Indent),
           $_writeans( Answer),
           write( ', which was '),
           $_show1( Found, Indent)
         ).

$_show( Answer, Indent) :-
         var( Answer),
         tab( Indent),
         $_writeans( 'empty goal').

$_show1( Derived from Answer, Indent) :-        /* Writes out the Found */
         (                                      /*   parts of an Answer */
           var( Derived),
           !,
           write( derived)
         ;
```

```
  write( Derived),
           write(' rule')
         ),
         (
  $_no_more_exp( Answer),
           nl
         ;
  write( ' from'),
           nl,
           Indent1 is Indent + 4,                /* Indent 4 more        */
           (
  instantiated(Answer),
             !,
             $_show( Answer, Indent1)
           ;
  $_show( 'an empty goal' was 'assumed false', Indent1)
           )
         ).

$_show1( Found, _) :-
         var(Found),
         !,
         write( derived),
         nl.

$_show1( Found, _) :-
         write( Found),
         nl.

$_writeans( Goal is true) :-
         !,
         write( Goal).
/*$_writeans( Answer) :-
         var( Answer),
         !,
         write( 'empty goal').
*/
$_writeans( Answer) :-
           write( Answer).

$_no_more_exp( Answer1 and Answer2) :-          /* Succeeds if all the  */
         !,                                    /*   terms in Answer    */
         Answer1 = ( _ is _ was no_exp),       /*   are no_exp terms   */
         $_no_more_exp( Answer2).
$_no_more_exp( Answer1 but Answer2) :-          /* Succeeds if all the  */
         !,                                    /*   terms in Answer    */
         Answer1 = ( _ is _ was no_exp),       /*   are no_exp terms   */
         $_no_more_exp( Answer2).
$_no_more_exp( Answer) :-
         Answer = ( _ is _ was no_exp).
:
```

Appendix 1A

Code Common to the Original B-Shell and the Modified B-Shell

The Prolog code contained in this section is used in conjunction with the Prolog code in Appendix 1 to provide the function of the Original B-Shell or with the code in Appendix 2 to provide the function of the Modified B-Shell

TRANSCRIPT FOR THE FIRST STUDENT

taken(706). taken(709). taken(720).
taken(744). taken(781). taken(809).
taken(782). taken(890,ai,2).
taken(890,ai,2). taken(739).
taken(745). taken(780).

TRANSCRIPT FOR THE SECOND STUDENT

taken(706). taken(709). taken(720).
taken(744). taken(781). taken(809).
taken(782). taken(890,ai,2).
taken(890,ai,2). taken(739).
taken(810). taken(890,networking,2).
taken(890,os,2). taken(890,os,2).

TRANSCRIPT FOR THE THIRD STUDENT

taken(706). taken(709). taken(720).
taken(744). taken(781). taken(809).
taken(782). taken(890,ai,2).
taken(890,ai,2). taken(739).
taken(810). taken(745).
taken(890,os,2).

```
/* Companion Prolog file for the thesis advisor knowledge file.
      Contains supporting Prolog code for initializations and
      non-trivial "pro" calls. This file is reconsulted when
      the main knowledge file is consulted or reconsulted.

      The two init routines (as specified in the main knowledge
      file) are called before chaining starts. The first of these
      loads a transcript a student transcript by asking the user
      for the name of a transcript file and then mapping the
      courses on that file into facts that are then asserted
      into working storage via the "postfact" command.
      The second init routine calculates the total number of
      credit hours taken by the student. This is done in an
      init routine instead of a "pro" routine because it is
      always needed and should not be done "on demand" in a rule.

      The single "pro" routine calculates the number of hours
      taken in seminars in a particular field. This is done
      in a pro routine instead of an init routine because most
      fields will have no seminars taken, and the number of hours
      should be computed only when relevant (i.e on demand).

      The only really kludgy thing is how facts are posted for
      seminars taken. The problem is that a student might (will)
      take more than one seminar for the same number of hours in
      the same field. When postfact is used to post that two
      such facts to working storage, the second identical fact
      will be treated as redundant by postfact and will not be
      posted. To fake postfact out, a third field is added to
      the fact in the form of a unique integer. This integer
      is ignored by the expert system, but it allows the
      "redundant" fact to be asserted.

      One final note is that the use of bagof is not efficient,
      but who cares on such a small problem? If lists being
      created by bagof were huge, it would be worth writing a
      routine that combined the behavior of bagof with that of
      postfact, thereby eliminating the need to create a list
      and then process it recursively. However, this problem
      is too small to worry about efficency.
*/
```

```
    /* ****** INIT ROUTINES ******** */

load_transcript :-              /* 1st "init" routine  */
        nl,
        write('What file has the student transcript? (same or filename) '),
        read(File),
        ( File == same,                     /* Use leftover working */
          !,                               /*  storage from last  */
          ( computed(_)                     /*  run of the system  */
          ;
            nl,
            write('No problem has been worked on before.'),
            nl
          )
        ;
          clearws,                         /* Reinitialize working */
          reconsult(File),                 /*  storage with new  */
          assert_courses                   /*  information       */
        ).

assert_courses :-                          /* Called by init rtne  */
        bagof('you took' Course,            /* Build list of course */
              taken(Course),                /*  numbers for 4-hour */
              Courses),                     /*  courses taken      */
        post_list(Courses),                /* Post facts for them  */
        bagof(                             /* Do same for seminars */
              'you took'(Field,Hours,N),    /* N needed to get a    */
              (taken(890,Field,Hours),      /*  unique fact when    */
                 increment_reg(sem_ctr,N)), /*  > 1 seminar taken  */
              Sems),                       /*  in same field      */
        post_list(Sems).

post_list([Head|Tail]) :-
        postfact(Head),                    /* Asserts fact if not  */
        post_list(Tail).                  /*  already there      */
post_list([]).


tot_cred :-                                /* 2nd "init" routine  */
        bagof(Course,
              computed('you took' Course),  /* Get list of 4-hour  */
              CList),                      /*  course numbers     */
        length(CList, NCourses),            /* Get how many of them */
        bagof(                             /* Get list of sem hours*/
              Hours,
              Field^N^computed('you took'(Field,Hours,N)),
              HList),
        sum_list(HList, SemTot),            /* Total seminar hours  */
        Total is (4 * NCourses) + SemTot,    /* Total course hours  */
        postfact('you have' Total hours).    /* Post in working stor */
```

```
sum_list([Head|Tail], Sum) :-
        sum_list(Tail, TailSum),
        Sum is TailSum + Head.
sum_list([],0).

/* ********************  P R O   R O U T I N E S  ********************* */

sem_tot(Field, TotHours) :-                    /* "pro" routine        */
        bagof(                                /* Get list of sem hours*/ ,
                Hours,                                /* in Field     */
                N^computed('you took'(Field, Hours, N)),
                HList),
        sum_list(HList, TotHours).             /* Total sem hrs in fld */
```

```
/* Handle askable goals and why questions.
       This is user interface stuff with some explanation thrown in.
*/

$_useranswer( Goal, Trace, Answer) :-          /* Driver call          */
/*      askable( Goal, _), {done in explore}    /* Is goal askable?     */
        $_freshcopy( Goal, Copy),              /* Make a copy to trash */
        $_useranswer( Goal, Copy, Trace,       /* Call the real thing  */
                Answer, 1).


$_useranswer( Goal, _, _, _, N) :-
        N > 1,                                 /* Don't ask again      */
        instantiated( Goal),                   /*   about an already-  */
        !,                                     /*   instantiated Goal  */
        fail.

$_useranswer( Goal, Copy, _. Answer, _) :-
        wastold( Copy, Answer, _),             /* Answer to Goal is    */
        $_instance_of( Copy, Goal),            /*   implied by a more  */
        !.                                     /*   general Goal       */

$_useranswer( Goal, _, _, true, N) :-
        wastold( Goal, true, M),               /* Get already-told     */
        M >= N.                                /*   sol'ns after Nth   */

$_useranswer( Goal, Copy, _, Answer, _) :-
        end_answers( Copy),                    /* It's all been said   */
        $_instance_of( Copy, Goal),            /*   before             */
        !,
        fail.

$_useranswer( Goal, _, Trace, Answer, N) :-    /* If all else fails    */
        $_askuser( Goal, Trace, Answer, N).    /*  ask user for sol'ns */


$_askuser( Goal, Trace, Answer, N) :-
        askable( Goal, ExternFormat),          /* Make sure it's askble*/
        $_format( Goal, ExternFormat, Question, /* Make English question*/
                [], Variables),
        $_ask( Goal, Question,                 /* Ask the question     */
                Variables, Trace, Answer, N).
```

```
$_ask( Goal, Question, Variables, Trace, Answer, N) :-
        nl,
        ( Variables = [],                       /* First time for this  */
          !,                                    /*   question           */
          write( 'Is it true:  ')
        ;
          write( 'Any (more) solution(s) to:  ') /* Try another sol'n   */
        ),
        write( Question),
        write( '? '),
        getreply( ynw, Reply),                  /* Get yes, no or why   */
        !,                                      /* Don't try this again */
        $_process( Reply, Goal, Question,       /* Handle the answer    */
                Variables, Trace, Answer, N).


$_process( why, Goal, Question,
                Variables, Trace, Answer, N) :-
        $_showtrace( Trace),                    /* Why -> explanation   */
        $_ask( Goal, Question,
                Variables, Trace, Answer, N).

$_process( yes, Goal, _,
                Variables, Trace, true, N) :-
        $_nextindex( Next),                     /* Yes -> new true      */
        Next1 is Next + 1,                      /*   thing to remember  */
        ( $_askvars( Variables),                /* Get values for vars  */
          assertz( wastold( Goal, true, Next))  /* Store as told fact   */
        ;                                       /* On backtracking,     */
          $_freshcopy( Goal, Copy),             /*   get a clean copy & */
          $_useranswer( Goal, Copy, Trace,      /*   ask for another    */
                Answer, Next1)                  /*   possible solution  */
        ).

$_process( no, Goal, _, _, _, false, N) :-
          $_freshcopy( Goal, Copy),
          wastold( Copy, true, _),              /* Ignore inconsistent  */
          !,                                    /*   information        */
          assertz( end_answers( Goal)),         /* Ask for no more      */
          fail
        ;
          $_nextindex( Next),                   /* Remember false info  */
          assertz( wastold(Goal, false, Next)), /*   as a told fact     */
          assertz( end_answers( Goal)).         /* Ask for no more soln */
                                                /*   (closed wld assmp) */
```

```
$_format( Var, Name, Name, Vars, [Var/Name|Vars]) :-
        var( Var),
        !.

$_format( Atom, Name, Atom, Vars, Vars) :-
        atomic( Atom),
        !,
        atomic( Name).

$_format( Goal, Form, Question, Vars0, Vars) :-
        Goal =..[Functor|Args1],
        Form =..[Functor|Forms],
        $_formatall( Args1, Forms, Args2, Vars0, Vars),
        Question =..[Functor|Args2].


$_formatall( [], [], [], Vars, Vars).

$_formatall( [X|XL], [F|FL], [Q|QL], Vars0, Vars) :-
        $_formatall( XL, FL, QL, Vars0, Vars1),
        $_format( X, F, Q, Vars1, Vars).


$_askvars( []).

$_askvars( [Variable/Name|Variables]) :-        /* Get values for all   */
        nl,                                     /*  variables in a list */
        write( Name),
        write(' = '),
        read( Variable),
        $_askvars( Variables).


$_showtrace( []) :-
        nl,
        nl,
        write( 'This was your question:'),
        nl.
```

```
$_showtrace( [Explan_structure|Trace]) :-
    Explan_structure = explan_structure(Text,Level,Silence,_,_),
    (
      Silence == n,
      write('Investigating whether to '),
      write(Text),
      ( Trace == []
        ;
        Trace \== [],
        write(' by, ')
      ),
      nl
    ;
      Silence == y
    ),
        nl,
/*
        write( 'To investigate '),
        write( Goal),
        write( ', by applying the '),
        write( Rule),
        write( ' rule.'),
*/
        $_showtrace( Trace).


$_instance_of( Term, Term1) :-                  /* Term is at least as  */
        $_freshcopy( Term1, Term2),             /*   general as Term1   */
        numbervars( Term2, 0, _),
        !,
        Term = Term2.                           /* Term & Term1 unify   */

$_freshcopy( Term, FreshTerm) :-                /* Keep from blitzing   */
        asserta( copy( Term)),                  /*   any variables in   */
        retract( copy( FreshTerm)),             /*   Term               */
        !.

lastindex( 0).                                  /* Initialize           */

$_nextindex( Next) :-                           /* Bump global counter  */
        retract( lastindex( Last)),
        !,
        Next is Last + 1,
        assert( lastindex( Next)).
```

```
/* Tools - These are things that you always forget to include
        but use frequently.
*/

/* Vi : Fake an on-the-fly modification of a program in a file
*/

:- op(900, fx, vi).                     /* Make 'vi' an operator        */

vi(File) :-                             /* Modify code in a File        */
        name(File,FString),                     /* Get File as String   */
        ViString = [118,105,32|FString],        /* Build "vi File"      */
        system(ViString),                       /* Do the vi command    */
        reconsult(File).                        /* Pull in modified code */

l :- system("l").                       /* Look at files in current dir */


/* List stuff   Right out of C&M
*/

member(X,[X|_]).                /* Is X a member of List, (p. 55)       */
member(X,[_|Rest]) :-
        member(X,Rest).

last(Last,[Last]).              /* Get last element of a list, (p. 149) */
last(Last,[_|Rest]) :-
        last(Last,Rest).

append([],L,L).                 /* Append 1st arg to 2nd, return in 3rd */
append([X|L1],L2,[X|L3]) :-
        append(L1,L2,L3).

rev(L1,L2) :-                   /* Reverse a list (p. 150)              */
        revzap(L1,[],L2).
revzap([X|L],L2,L3) :-
        revzap(L,[X|L2],L3).
revzap([],L,L).

insort([H|T],S,O) :-            /* Insertion sort from pp. 156          */
        insort(T,L,O),
        plugin(H,L,S,O).
insort([],[],_).
plugin(Elem,[Elem2|T],[Elem2|L],a) :-
        Elem2 @< Elem,                  /* Lexical ordering handles all */
        !,
        plugin(Elem,T,L,a).
```

Appendix 2

Modified B-Shell Code

Code unique to the modified B-shell. This code used in combination with the common code in Appendix 1A provides the function of the modified B-Shell.

```
/*  Top-level driving procedure
        After getting a high level goal to justify or reject,
        look first for a positive answer, and if one is found
        prohibit negative answers from being found (i.e. if I
        can prove it true, I can't also prove it false).
        If no positive answer is found, then look for negative
        answer(s) to report to show why it is false.
        If no answers can be found at all, report that and quit.
*/

expert :-
        ( _ : _                         /* If a knowledge base has not  */
        ;                               /*   been loaded yet, then      */
          loadkb                        /*   ask the user for one       */
        ),
        $_getquestion( Question),       /* Get something to work on     */
        create_top_goal(Question),      /* store the top level goal in  */
                                        /* data base in text form for   */
                                        /* use in explanations.         */
        $_initialize,                   /* KB Init Prolog routine       */
        ( $_answeryes( Question)        /* Try to find yes answer       */
        ;
          $_answerno( Question)         /* O/W look for no answer       */
        ;
          nl,
          write('No more solutions could be found.'),
          nl
        ).


$_answeryes( Question) :-
        $_markstatus( negative),        /* No positive answer yet       */
        $_explore( Question, [], Answer),/* Look with trace empty       */
        $_positive( Answer),            /* Accept only positive ans     */
        $_markstatus( positive),        /* Positive answer found        */
        present_explan( Answer),                /* Display the answer  */
        nl,
        write( 'More solutions? '),     /* Look for more solns?         */
        getreply( ynw, Reply),          /* Get yes or no from user      */
        Reply == no.                    /* yes backtrack to explore     */

$_answerno( Question) :-
        retract(no_positive_answer_yet), /* No positive answer yet so   */
        !,                               /*    can have negative answer */
        $_explore( Question, [], Answer),/* Look with trace empty       */
        $_negative( Answer),            /* Accept only negative ans     */
        present_explan(Answer),         /* Display the answer           */
        nl,
        write('More negative solutions? '), /* Look for more solns?     */
        getreply( ynw, Reply),             /* Get yes or no from user   */
        Reply == no.                       /* yes backtrack to explore */


$_markstatus( negative) :-
```

```
        assert(no_positive_answer_yet).  /* Set bit until pos ans found */

$_markstatus( positive) :-
          retract(no_positive_answer_yet), /* Reset when pos ans found  */
          !
        ;
          true.                  /* Succeed if pos answer already found  */


$_getquestion( Question) :-        /* Get question from user        */
        ( questform(_),
          nl,
          write('Ask one of the following questions:'),
          nl,
          !,
          $_write_quests
        ;
          true
        ),
        nl,
        write('Question, please:'),
        nl,
        read( Question).

$_write_quests :-
        questform(Quest),
        write(Quest),
        nl,
        fail.
$_write_quests.


$_initialize :-
        init(X),
        $_run_init(X),
        fail.
$_initialize.

$_run_init([First|Rest]) :-
        !,
        call(First),
        !,
        $_run_init(Rest).
$_run_init([]) :-
        !.
$_run_init(Thing) :-
        call(Thing),
        !.


loadkb :-
        nl,
        write('What knowledge base file do you want to load? '),
        read(File),
        ( _ : _.
```

```
        !,
        write('Replace or add to the existing knowledge base? (r, a) '),
        read(Ans),
        ( Ans == a,
          consult( File)
        ;
          reconsult( File)
        )
      ;
        reconsult( File)
      ).

clearws :-
        abolish( wastold, 3),
        abolish( end_answers, 1),
        abolish( lastindex, 1),
        assert( lastindex( 0)),
        abolish( false_goal, 1),
        abolish( true_goal, 1),
        abolish( computed, 1),
/*      abolish( derived, 1),*/
        abolish( no_positive_answer_yet, 0).

dumpws :-
        listing( lastindex),
        listing( end_answers),
        listing( wastold),
        listing( computed),
/*      listing( derived),*/
        listing( false_goal),
        listing( true_goal).

traceon :-
        trace_flag,
        write( 'Chaining is already being traced.'),
        nl,
        !.
traceon :-
        assert( trace_flag),
        write( 'Chaining will be traced.'),
        nl.

traceoff :-
        trace_flag,
        abolish( trace_flag, 0),
        write( 'Chaining will not be traced.'),
        nl,
        !.
traceoff :-
        write( 'Chaining was not being traced.'),
        nl.
```

```
/* This version of explore is a modification and extension
        of the Bratko version from p. 331.  The modifications
        are chiefly performance enhancements to avoid some of
        the redundant recursive calls to explore and to place
        cuts where they can limit the clauses that apply to a
        given goal.

        The extensions include a number of additional operators
        to handle more interesting rule constructions and new
        hidden fact type to allow a pseudo-forward chaining.
        The new operators are:

                no Goal : Universal negation, Goal always fails
                        forall(X, ~f(X))

                is_no Goal : Existential negation, Goal fails once
                        exists(X, ~f(X))

                Goal1 xor Goal2 : Exclusive or, Or with a cut
                        (Goal1, ! ; Goal2)

                (pro Goal) : Prolog callable Goal, Meta call
                        call(Goal), !

                computed(Goal) : Fact asserted via "forward chaining"


        The reason "no" and "is_no" were chosen as negation
        operators was to avoid ambiguity with Prolog's "not"
        operator.  Of the two, "no" behaves most like Prolog's
        "not" in that it only succeeds if Goal cannot succeed.
        The "is_no" operator fails only if the goal cannot fail.

        The "xor" operator complements the original inclusive or
        and behaves like an or-with-a-cut to let the user cut down
        on backtracking when the first solution in a disjunction
        is OK.

        For Prolog callable goals, the goal should be enclosed
        in parentheses to insure that the precedences of the
        built-in Prolog operators are maintained.
        The reason the "pro" operator exists, instead of just
        sending goals to Prolog automatically, is to avoid
        ambiguity with "useranswer" goals.

        A "computed" fact is presumably asserted by a prolog rule
        called via the "pro" operator.  This allows some
        forward-chaining-like behavior by putting facts into a
        working storage area "invisibly."  The format of these
        computed facts is ugly:

                computed(Fact).
```

```
        rather than the prettier user-defined forms.  This is
        because these facts are part of working storage, not
        the knowledge base, and they need not be explained.
        Newly computed facts are put into the knowledge base via
        postfact(Fact), which asserts a computed(Fact) clause
        only if a unifiable one does not already exist.
*/

/* explore( Goal, Trace, Answer).

        Explore a Goal to arrive at an Answer considering the Trace
        of previously answered goals.

        Can be done if Goal is a fact, the consequent of a rule,
        a conjunction of subgoals, a disjunction of alternative
        goals, a negated goal, a Prolog-callable "meta-goal"
        or a user-answereable "question" goal.
*/

$_explore( Goal1 and Goal2, Trace, Answer) :-   /* Conjunctive goal     */
        $_exp_trace( 'Enter and ', Goal1 and Goal2),
        !,
        $_explore( Goal1, Trace, Answer1),      /* Do first term        */

        $_continue( Answer1, Goal1 and Goal2,   /* Handle the rest      */
                Trace, Answer).

$_explore( Goal1 xor Goal2, Trace, Answer) :-   /* Exclusive or behaves */
        $_exp_trace( 'Enter xor ', Goal1 xor Goal2),
        !,                                      /*   like an or with a  */
        (                                       /*   cut to avoid bktk  */
          $_explore( Goal1, Trace, Answer),     /* Explore first one    */
          $_positive( Answer),                  /* If it's true, done   */
          !
        ;
          $_explore( Goal2, Trace, Answer),     /* O/W try other side   */
          $_positive( Answer),                  /* If it's true, done   */
          !
        ;
          !,                                    /* O/W call the false   */
          $_false_or(Goal1, Goal2, Trace, Answer)/*   handler           */
        ).

$_explore( Goal1 or Goal2, Trace, Answer) :-    /* Inclusive "or" is ;  */
        $_exp_trace( 'Enter or ', Goal1 or Goal2),
        !,
        (
          $_explore( Goal1, Trace, Answer),     /* Explore first one    */
          $_positive( Answer),                  /* If it's true, done   */
          $_set_true( Goal1)

        ;
          $_explore( Goal2, Trace, Answer),     /* O/W try other side   */
          $_positive( Answer),                  /* If it's true, done   */
```

```
          $_set_true( Goal2)

        ;                                       /* O/W it's false       */
          not true_goal( Goal1),                /* Be sure no positive  */
          not true_goal( Goal2),                /*   answers were found */
          !,
          $_false_or(Goal1, Goal2, Trace, Answer)
        ).

$_explore( no Goal, Trace, Answer) :-           /* Closed world negate  */
        $_exp_trace( 'Enter no ', Goal),
        !,                                      /*   all(X, not X)      */
        (
          $_explore( Goal, Trace, RawAnswer1),  /* Explore the Goal     */
          $_positive(RawAnswer1),               /* If there's any way   */
          !,                                    /*   it's true, then    */
          $_invert( no, RawAnswer1, Answer)     /*   return a false     */
        ;
          explainable( Goal, Form),             /* O/W try to build a   */
          $_format( Goal, Form, Phrase,[],Vars),/*   nice explanation   */
          !,                                    /*   with instantiated  */
          Answer =                              /*   variables          */
            explan_structure(no Phrase was 'not disproved',
                             1,n,concept0,false)
        ;
          !,
          Answer =                              /* O/W build an ugly    */
                                                  /* explanation        */
            explan_structure(no Goal is true was 'not disproved',
                             1,n,concept0,false)
        ).

$_explore( is_no Goal, Trace, Answer) :-        /* Open world negation  */
        $_exp_trace( 'Enter is_no ', Goal),
        !,                                      /*   exists(X, not X)   */
        (
          $_explore( Goal, Trace, RawAnswer1),  /* If find a negative   */

          $_negative( RawAnswer1),              /*   answer, then there */
          $_set_false( Goal),                   /*   exists a Goal that */
          $_invert(is_no, RawAnswer1, Answer)   /*   is false           */
        ;
          not false_goal( Goal),                /* If no false answers  */
          !,                                    /*   can be found, then */
          $_explore( Goal, Trace, RawAnswer1),  /*   get a true answer  */

          $_positive( RawAnswer1),              /*   and invert it      */
          $_invert(is_no, RawAnswer1, Answer)
        ).
/*-------------------------------------------------------------------*/
/* the last five $_explore clauses may be considered the 'primatives' of*/
/* shell, that is, the non-compound conditions that return text for the */
/* eventual explanation.   These clauses access get_explan to create the*/
/* structures used for explanations.                                    */
/*-------------------------------------------------------------------*/
```

```
/*-----------------------------------------------------------------*/
/* prolog callable goals are dealt with here.   Since this could be any-*/
/* thing the explanation will be crude, simply returning the Goal as the*/
/* text item in the explanation.    Silence is assumed 'n' however the  */
/* user level is set at 10 to restrict inclusion in the explanation     */
/* except for the very curious that pick such a detailed level.         */
/*-----------------------------------------------------------------*/
$_explore( pro Goal, Trace,
                Pro_answer was called) :-       /* Prolog-provable goal */
        $_exp_trace( 'Enter pro ', Goal),
        !,
        (
          call( Goal),                          /* Send Goal to prolog  */
          !,                                    /* No backtracking cuz  */
          Truth = true,                         /*   it might fail      */
          Pro_answer = explan_structure(Goal,10,/* assume user level 10 */
                       n,concept0,true)         /* not silent, catch all*/
                                                /* concept number 0.    */
        ;                                       /* If call fails return */
          Truth = false,                        /*   false, don't fail  */
          Not_goal = Goal 'evaluated to false', /* Negage answer.       */
          Pro_answer = explan_structure(        /* Assume user level 10 */
                       Not_goal,10,n,concept0,  /* not silent, catch all*/
                       false)
                                                /* concept number 0.    */
        ).
/*-----------------------------------------------------------------*/
/* FACTS are dealt with here.   Facts have explainable clauses and the  */
/* structure can be obtained through get_explan.   The truth value by   */
/* default is 'true' so the pro_text is obtained.  Con_text is only     */
/* obtained later if there is a need to negate the fact.                */
/*-----------------------------------------------------------------*/

$_explore( Goal, Trace, Explan_structure was 'given as fact') :-
                                                /* Goal is a fact       */

        fact : Goal,
        get_explan(Goal,true,Explan_structure),
        $_exp_trace( 'Found fact ', Goal).
/*-----------------------------------------------------------------*/
/* Computed facts:  clause postfact asserts computed() clauses to data  */
/* base.   Structures may be obtained through get_explan.   Truth value */
/* by default is 'true' so the pro_text is obtained.  Con_text is only  */
/* obtained later if there is a need to negate the fact.                */
/*-----------------------------------------------------------------*/

$_explore( Goal, Trace, Explan_structure was computed) :-
                                                /* Goal was computed &  */
                                                /*   asserted by prolog */
        computed(Goal),                         /*   using postfact     */
        get_explan(Goal,true,Explan_structure),
        $_exp_trace( 'Found computed ', Goal).
/*-----------------------------------------------------------------*/
/* Rules:  Rules are not much different than explaining anything else   */
```

```
/* except that they tend to be statements of more general concepts.  The*/
/* pro or con text is available via get_explan, and the determination of*/
/* which is driven by TruthValue.   The actual explanation only needs   */
/* Goal and Answer when presented however the added verbage is evaluable*/
/* and remains to preserve proper operation of the system.   The extra  */
/* items are ignored when presenting the explanation.  A typical example*/
/* from the previous systems explanation would be "doa thesis in os is  */
/* false, which was derived by thesis_in_field rule from ( supporting   */
/* and contradictory support followed but is not evaluated here )"  This*/
/* will now be presented as "do not do a thesis in os, which is         */
/* supported by ( supporting and contradictory information follow )"    */
/*                                                                      */
/*     One other item to be concerned with here is the maintenance of   */
/* the rule trace (Trace).   This must also be formatted to a readable  */
/* form (create explan_structure's) for intermediate 'why' questions.   */
/*----------------------------------------------------------------------*/

 $_explore( RawGoal, Trace,                      /* Goal is consequent   */
    Goal was 'derived by a rule' from Answer) :- /*   of a rule          */

        Rule : if Condition then RawGoal,
        $_exp_trace( Rule, ' rule ', RawGal),
        get_explan(RawGoal,true,Goal_for_trace),/* create explan. struct*/
        $_explore( Condition,                   /* Satisfy Condition    */
                [Goal_for_trace  | Trace],      /* Set up for why quest */
                Answer),
        $_truth( Answer, TruthValue),
        get_explan(RawGoal,TruthValue,Goal).    /* create explan. struct*/

/*----------------------------------------------------------------------*/
/* user supplied answers return Goal formatted with information supplied*/
/* which is then fed to get_explan as an Id to get the final explanation*/
/* structure desired.   'Answer' for $_userans is either true or false. */
/*----------------------------------------------------------------------*/

$_explore( Goal, Trace,GoalStructure was told) :- /* User-askable goal    */
        askable( Goal, _),
        $_exp_trace( 'Enter askable ', Goal),
        !,                                      /* Can only ask 1 way   */
        $_useranswer( Goal, Trace, Answer),     /* Goal explan_structure*/
        get_explan(Goal,Answer,GoalStructure).  /* created by get_explan*/



$_continue( Answer1, Goal1 and Goal2,           /* Rest of conjunction  */
                Trace, Answer) :-
        $_positive( Answer1),                   /* First part OK        */
        $_explore( Goal2, Trace, Answer2),      /* Explore the rest     */
        ( $_positive(Answer2),                  /* If rest is OK...     */
          Answer = Answer1 and Answer2           /* ...return conjunc    */
        ;
          $_negative(Answer2),                  /* O/W return answer    */
          ( Answer2   A21 but A22,
            Answer = Answer1 and Answer2        /* ...return conjunc    */
          ;
```

```
            Answer = Answer1 but Answer2        /* ...return conjunc    */
          )
/*        Answer = Answer2                     /* from rest            */
        ).

$_continue( Answer1, Goal1 and Goal2,          /* If couldn't get yes  */
                _, Answer1) :-                 /*   or no return       */
        $_negative( Answer1).                  /*   a no answer        */


$_false_or(Goal1, Goal2, Trace, Answer) :-     /* Handles false "or"s  */
        $_get_neg_ans( Goal1, Trace, Answer1),
        $_get_neg_ans( Goal2, Trace, Answer2),
        Answer = Answer1 and Answer2.           /* Return both falses   */

$_get_neg_ans( Goal, Trace, Answer) :-
        $_explore( Goal, Trace, RawAnswer),    /* Explore first term   */
        $_negative(RawAnswer),                 /*   for a false answer */
        $_set_false( Goal).


$_get_neg_ans( Goal, Trace, Answer was negated) :-
        not false_goal( Goal),
        !,
        get_explan(Goal,false,Answer).


/*$_truth(Var, false) :-                       /* Just in case...      */
/*      var(Var),
        Var = false,
        !.
*/

$_truth(explan_structure(_,_,_,_,TruthValue) was _ , TruthValue) :- !.
                                               /* Gets truth value.    */

$_truth( Answer1 and Answer2, TruthValue) :-
          $_truth( Answer1, true),
          $_truth( Answer2, true),      /* If both sides are true...    */
          !,
          TruthValue = true             /* ...then the whole thing is   */
        ;
          TruthValue = false.           /* O/W it's false               */

$_truth( Answer1 but Answer2, TruthValue) :-
          $_truth( Answer1, true),
          $_truth( Answer2, true),      /* If both sides are true...    */
          !,
          TruthValue = true             /* ...then the whole thing is   */
        ;
          TruthValue = false.           /* O/W it's false               */


$_positive( Answer) :-
        $_truth( Answer, true).
```

```
$_negative( Answer) :-
        $_truth( Answer, false).


$_set_true( Goal) :-
            true_goal( Goal),
            !
          ;
            assert( true_goal(Goal)).

$_set_false( Goal) :-
            false_goal( Goal),
            !
          ;
            assert( false_goal(Goal)).
/*-------------------------------------------------------------------*/
/* inverting from a positive to negative requires a little backtrack- */
/* ing ( not in the prolog sense ).   The original explan_frame must  */
/* be located and the Id recreated, now instantiated with the infor-  */
/* mation that was avaiable at the time the positive explanation was  */
/* created.   Once obtained the negative explanation is created by the*/
/* normal application of get_explan.    The converse must also be     */
/* available, hence the second clause.                                */
/*-------------------------------------------------------------------*/
$_invert(_,Pos_explan was Found, Neg_explan was Found) :-
         Pos_explan   explan_structure(Pos_text,_,_,_,_),
         explan_frame(id(Id),_,_,_,_,pro_text(Pos_text),_),
         get_explan(Id,false,Neg_explan).

$_invert(_,Neg_explan was Found, Pos_explan was Found) :-
         Neg_explan   explan_structure(Negt_text,_,_,_,_),
         explan_frame(id(Id),_,_,_,_,_,con_text(Neg_text)),
         get_explan(Id,true,Pos_explan).



$_format_goal( RawGoal, FormGoal) :-
          explainable( RawGoal, Form),
          $_format( RawGoal, Form, FormGoal,[],Vars),
          !.
$_format_goal( Goal, Goal).

$_exp_trace( Heading, Goal) :-
          trace_flag,
          write( Heading),
          write( Goal),
          nl,
          !
        ;
          true.

$_exp_trace( Head1, Head2, Goal) :-
          trace_flag,
          write( Head1),
```

```
    write( Head2),
    write( Goal),
    nl,
    !
;
    true.
```

```
/*-------------------------------------------------------*/
/* create-top-goal:  determine the general category of   */
/* question asked by the user and convert it to a state- */
/* ment indicating the top level goal.   Provide default */
/* information as needed and assert top_goal(Top_Goal) in */
/* the deta base for use during explanation.             */
/*-------------------------------------------------------*/
:- op(100,fx,['determine whether to do a']).

explainable('determine whether to do a' _ in _, 'determine whether to do a'
            'thesis or project' in 'the students chosen specialty area.').

explainable('determine whether to do a' _, 'determine whether to do a'
             'project or thesis.').


/*-------------------------------------------------------*/
/* create a top goal statement based on the Question asked */
/*   The first half looks for a question of the form:    */
/*      doa What in Which, examples are "doa project in ai"*/
/*      "doa What in os" and the like.   The second half  */
/*   looks for the form of   doa What, e.g. doa project or */
/*   doa thesis.     In both cases the given information is*/
/*   converted to a new format, that of a statement.     */
/*   The user may have left one or more elements uninstan- */
/*   tiated so $_format_goal instantiates them to default */
/*   values.   The explainable clauses above provide the  */
/*   default text for the two types of questions.   If    */
/*   neither type of question is asked then return the    */
/*   question unchanged.                                 */
/*-------------------------------------------------------*/

create_top_goal(Question) :-
     (
      Question = doa What in Which,
     Top_goal = 'determine whether to do a' What in
                Which,
     $_format_goal(Top_goal,Frmt_Top_Goal),
     asserta(top_goal(Frmt_Top_Goal))
    )
    ;
     (
      Question = doa What,
     Top_goal = 'determine whether to do a' What,
     $_format_goal(Top_goal,Frmt_Top_Goal),
     asserta(top_goal(Frmt_Top_Goal))
     )
     ;  /* neither of the expected question forms */
        /* return it unchanged */
     (
     asserta(top_goal(Question))
     ).
```

```
/*-------------------------------------------------------------*/
/* routines responsible for generation of new line character if */
/* text being printed is going to go past the edge of the page. */
/* Two routines exist.  1) Normal test during printing to see if*/
/* the next string will result in passing 78 characters in      */
/* width, if it will "nl" is sent and the count set to the      */
/* length of the new string about to be printed on the next line*/
/* 2) if the program generates "nl" then the length must be     */
/* reset.    The total length is maintained in the data base for*/
/* ease of access.                                              */
/*-------------------------------------------------------------*/

 new_line(String) :-
    total_length(Total_length),
    atomic(String),
    name(String,String_list),
    string_length(String_list,Next_length),
    New_length is Total_length + Next_length,
    (
      New_length > 78,
      nl,
      retract(total_length(Total_length)),
      asserta(total_length(Next_length))
      ;
      retract(total_length(Total_length)),
      asserta(total_length(New_length))
    ).

 new_line(String) :-  /* in case total_length not existing yet */
    not total_length(_),
    atomic(String),
    name(String,String_list),
    string_length(String_list,Next_length),
    asserta(total_length(Next_length)).

/* HANDLE NON-ATOMIC ENTITIES */

 new_line(String) :-
    total_length(Total_length),
    not atomic(String),
    String =.. Parts,
    parts_length(Parts,Next_length),
    New_length is Total_length + Next_length,
    (
      New_length > 78,
      nl,
      retract(total_length(Total_length)),
      asserta(total_length(Next_length))
      ;
      retract(total_length(Total_length)),
      asserta(total_length(New_length))
    ).
```

```
new_line(String) :-  /* in case total_length not existing yet */
   not total_length(_),
   not atomic(String),
   String =.. Parts,
   parts_length(Parts,Next_length),
   asserta(total_length(Next_length)).

/* reset line length */

reset_line_length(New_length) :-
     total_length(Total_length),
     retract(total_length(Total_length)),
     asserta(total_length(New_length)).

reset_line_length(New_length) :-
     not total_length(Total_length),
     asserta(total_length(New_length)).

string_length([],0) :- !.

string_length([Head | Tail], Length) :-
   string_length(Tail, Length2),
   Length is Length2 + 1.

parts_length([], 0) :- !.

parts_length([Head | Tail], Length) :-
   atomic(Head),
   name(Head,HeadChars),
   string_length(HeadChars,L1),
   parts_length(Tail,L2),
   Length is L1 + L2.

parts_length([Head|Tail],Length) :-
   not atomic(Head),
   Head =.. HeadList,
   parts_length(HeadList,L1),
   parts_length(Tail,L2),
   Length is L1 + L2.
```

```
/* Knowledge file that recommends whether to do a thesis or a project
       and suggests areas in which to do the thesis or project.
       The rules are intended to illustrate the construction of a
       knowledge file, not to give sound advice to students.

       A companion file containing supporting Prolog code is called
       "advpro" and should be looked at to discover how some of
       the facts make it into working storage.
*/

/* ********** OPERATORS for defining conceptual structures ************ */

:- op( 100, xfx, ['concentration is', 'concentration needs'] ).
:- op( 100, xfx, ['second course is', 'seminars totaled'] ).
:- op( 100, fx, ['you took', 'you have', 'your specialty is'] ).
:- op( 100, fx, [doa, 'core area'] ).
:- op( 90, xfx, in).                          /* Lower number for a   */
:- op( 90, xf, hours).                        /*   subtree            */


/* ************************** R U L E S ***************************** */

thesis_in_field : if                        /* Major goal for thesis      */
       'you finished the core' and          /* Simple subgoal             */
       'you have' N hours and               /* parse > 'you have'(hours(N)) */
       pro (N >= 44) and                    /* Use Prolog for test        */
       'your specialty is' Field and        /* Ask user for concentration */
       major 'concentration is' Field       /* Another simple subgoal     */
    then
       doa thesis in Field.                 /* parse> doa(in(thesis,Field)) */

thesis : if
       doa thesis in Field                  /* Map more complex goal into */
    then
       doa thesis.                          /*   a simpler goal           */


project_in_field : if                                 /* Major project goal */
       'you finished the core' and
       ( 'your specialty is' nothing                  /* Either state it    */
       xor                                            /*   explicitly or    */
         no 'your specialty is' Anything              /*   infer implicitly */
       ) and                                          /* xor freezes answer */
       'you have' N hours and
       pro (N >= 46) and
       major 'concentration is' MaField and
       'core area' MiField and                        /* Get a second Field */
       pro (MaField \== MiField) and                  /*   that's different */
       minor 'concentration is' MiField and           /* See if it's a minor */
       ( pro (Field = MaField)                        /* Retrun one and then */
              or                                      /*   the other on back- */
                pro (Field - MiField)                 /*   tracking for     */
              )                                       /*   alternate solution */
    then
```

```
        doa project in Field.

project : if
        doa project in Field
    then
        doa project.


core_done : if
        'you took' 706 and
        'you took' 709 and
        'you took' 720 and
        'you took' 744 and
        'you took' 781 and
        'you took' 809
    then
        'you finished the core'.

concentration : if
        'core area' Field and                          /* Find a field */
        Field 'second course is' Course and            /* Get 2nd crse */
        'you took' Course and                          /* Took it?     */
        Field 'seminars totaled' N hours and           /* Seminar hrs  */
        Level 'concentration needs' N hours            /* Hrs => Level */
    then
        Level 'concentration is' Field.

sem_hour : if
        pro (sem_tot(Field,N))                 /* Prolog routine does  */
    then                                       /*   the arithmetic     */
        Field 'seminars totaled' N hours.

c_level : if                                   /* Rule with 2 clauses  */
        pro (N >= 4)
    then
        major 'concentration needs' N hours.

c_level : if                                   /* Second clause        */
        pro (N >= 2)
    then
        minor 'concentration needs' N hours.


/* ************************  F A C T S  ************************** */

fact : 'core area' Field :-                    /* Specify core areas   */
        member( Field,
                [ai, os, theory, languages, networking, hardware]).

fact : ai 'second course is' 782.              /* Specify 2nd courses  */
fact : os 'second course is' 810.              /*   in core areas      */
fact : theory 'second course is' 850.
fact : languages 'second course is' 711.
fact : networking 'second course is' 745.
fact : hardware 'second course is' 721.
```

```
/* ************************ INTERFACE stuff ************************** */

askable( 'your specialty is'  _ ,              /* Concentration area   */
        'your specialty is'                    /*   will be entered    */
        'Which concentration area').           /*   by the user        */

questform( 'doa What in Which.').              /* Sample questions     */
questform( 'doa thesis in What.').             /*   users can ask      */
questform( 'doa project.').
questform( 'doa What in os.').
questform( 'doa What.').


/* ************************ INITIALIZATIONS ************************** */

                                               /* Routines to run      */
init( [load_transcript, tot_cred] ).           /*   before chaining    */

:- reconsult( advpro ).                        /* Link Prolog code     */
                                               /*   to knowledge base  */
```

```
:- op( 900, xfx, :).
:- op( 880, xfx, then).
:- op( 870,  fx, if).
:- op( 800, xfx, was).
:- op( 600, xfx, from).
:- op( 600, xfx, by).
:- op( 570, xfy, or).
:- op( 570, xfy, xor).
:- op( 550, xfy, and).
:- op( 550, xfy, but).
:- op( 540,  fy, silent).
:- op( 530,  fy, no).
:- op( 530,  fy, is_no).
:- op( 520,  fx, pro).
:- op( 300,  fx, 'derived by').

:- op( 100,  fx, 'your').
:- op( 100,  fx, ['you have', 'your specialty is', 'you did not take']).
:- op( 100,  fx, ['you do not have', 'your specialty is not']).
:- op( 100,  fx, ['do a thesis in'. 'do not do a thesis in']).
:- op( 100,  fx, [doa, 'core area']).
:- op( 100,  fx, ['do a project in', 'do not do a project in']).
:- op( 100,  fx, 'you have a core in').
:- op( 100,  fx, 'you took').
:- op(  90, xfx, in).

:- op(  85,  xf, 'seminar hours').
:- op(  85, xfx, 'concentration is not').
:- op(  85, xfx, ['concentration is', 'concentration needs']).
:- op(  85, xfx, 'concentration is').
:- op(  85, xfx, ['second course is', 'seminars totaled']).
:- op(  85,  xf, 'hours of courses').

:- op(  80,  xf, 'was not in your core').
:- op(  80,  xf, hours).
:- op(  80,  xf, 'your field').
:- op(  80,  xf, 'evaluated to_false').
:- op(  80,  xf, 'hours of courses').
```

```
/*-------------------------------------------------------*/
/* each pro_text and con_text portion needs an 'explainable'*/
/* clause to supply missing information of the user or    */
/* inference has not supplied it.  This is done by use of  */
/* $_format_goal and associated clauses during conversion of*/
/* the text from native form to sentence form in get_explan */
/*-------------------------------------------------------*/
/* thesis in field explanations */
 explainable('do a thesis in' Field,'do a thesis in' 'your field').
 explainable('do not do a thesis in' Field,'do not do a thesis in' 'your field').

/* project in field explanations */
 explainable('do a project in' Field, 'do a project in' 'your field').
 explainable('do not do a project in' Field,'do not do a project in' 'your field').

/* concentration explanations */
 explainable('your' Level 'concentration is' Field,'your' 'major or minor'
             'concentration is' 'not known').
 explainable('your' Level 'concentration is not' Field,'your' 'major or minor'
             'concentration is not' 'known').

/* seminar hours explanations */
 explainable('you have' N 'seminar hours' in Field,'you have' some
             'seminar hours' in 'some field').
 explainable('you do not have' N 'seminar hours' in Field,'you do not have' some
             'seminar hours' in 'some field').

/* sepecialty explanations */
 explainable('your specialty is' Field, 'your specialty is' 'in some field').
 explainable('your specialty is not' Field,'your specialty is not'
             'in some field').

/* Have N hours explanations */
 explainable('you have' N 'hours of courses','you have'
             'an undetermined number of' 'hours of courses').
 explainable('you do not have' N 'hours of courses','you have'
             'an undetermined number of' 'hours of courses').

/* core area explanations */
 explainable('you have a core' in Field,'you have a core' in 'a core area').
 explainable(Field 'was not in your core','the course' 'was not in your core').

/* you took explanations */
 explainable('you took' Course, 'you took' 'a course').
 explainable('you did not take' Course,'you did not take' 'a course').

/*-------------------------------------------------------*/
/* other explan_frames have default text that does not require */
/* sustitution therefore they do not appear here.          */
/*-------------------------------------------------------*/
```

```
/*-----------------------------------------------------------------*/
/* Concepts for summarization.  Summarization is a level option    */
/* when the user selects the user_level after a consultation has   */
/* reached a conclusion.                                           */
/*-                                                                */
/* structures are of the form:                                     */
/*      concept_text(Concept_id,Line,Text).                        */
/*                                                                 */
/*  where Concept_id is the identification of the concept and is the*/
/* same id as found in the ako( ) portion of the explan_frame's    */
/*-----------------------------------------------------------------*/

concept_text(concept1,1,' Doing a thesis is the preferred choice if you').
concept_text(concept1,2,' have completed the core courses with at least').
concept_text(concept1,3,' 44 hours and specialized in a core area.  The').
concept_text(concept1,4,' project option requires major and minor fields').
concept_text(concept1,5,' whereas this option only requires the major field.').

concept_text(concept2,1,' Doing a thesis is one of two options to complete').
concept_text(concept2,2,' the requirements for a Masters degree, the other').
concept_text(concept2,3,' is a project.').

concept_text(concept3,1,' Doing a project is one of two options to complete').
concept_text(concept3,2,' the requirements for a Masters degree, the other').
concept_text(concept3,3,' is a thesis.').

concept_text(concept4,1,' Doing a project is the preferred choice if you').
concept_text(concept4,2,' have not chosen a specialty and you fulfill the').
concept_text(concept4,3,' requirements of at least 46 credit hours, a').
concept_text(concept4,4,' major concentration, and a minor concentration.').

concept_text(concept5,1,' The core courses consist of courses 706, 709,').
concept_text(concept5,2,' 720, 744, 781, and 809.  Completion of the').
concept_text(concept5,3,' core is required for all Masters Candidates.').

concept_text(concept6,1,' Determination of a concentration is done by').
concept_text(concept6,2,' examining courses to see if a core area course').
concept_text(concept6,3,' and the complement course for that area were').
concept_text(concept6,4,' completed {the number of credit hours').
concept_text(concept6,5,' determines if this is a major or minor}.').

concept_text(concept7,1,' Seminar hours are used in determining possible').
concept_text(concept7,2,' concentration areas as they may be a valid').
concept_text(concept7,3,' second course.').

concept_text(concept8,1,' A major concentration needs at least 4 credit').
concept_text(concept8,2,' hours.').

concept_text(concept9,1,' A minor concentration needs at least 2 credit').
concept_text(concept9,2,' hours.').

concept_text(concept10,1,' If you have a specialty you should be doing a').
concept_text(concept10,2,' thesis in your specialty area.').
```

```
concept_text(concept11,1,' A minimum number of hours are required for both').
concept_text(concept11,2,' options, this is calculated from your').
concept_text(concept11,3,' transcripts.').

concept_text(concept12,1,' Taking courses in a core area can satisfy the').
concept_text(concept12,2,' concentration requirements.').


concept_text(concept15,1,' Each core area {ai, os, theory, languages,').
concept_text(concept15,2,' networking, and hardware} course has a ').
concept_text(concept15,3,' complementary course to make it a concentration.').
```

```
/*-------------------------------------------------------*/
/* basic frame structures that contain knowledge and infor-*/
/* mation that is used to construct explanation for the   */
/* shell.                                                 */
/* -     - -                                              */
/* explan_frame          Identifies this structure.       */
/*                                                        */
/* id                    Used to match frame to rule, fact,*/
/*                       or user supplied information.    */
/*                                                        */
/* ako                   A-Kind-of, used to link to general*/
/*                       concept for summarization.       */
/*                                                        */
/* user_level            User modeling will restrict some */
/*                       information being shown to user, */
/*                       values range from 1 through 10 and*/
/*                       explanation component will perform*/
/*                       a greater than or equal test to  */
/*                       determine if text is shown.      */
/*                                                        */
/* silence               Legal values are y,n, and always. */
/*                       'always' indicates that the text */
/*                       is always shown regardless of the */
/*                       values of user_level and ancestor */
/*                       nodes that may have silence set to*/
/*                       'y '                             */
/*                       If 'y' then all subtrees are     */
/*                       suppressed unless and 'always' is */
/*                       found in the subtree(s)          */
/*                       If 'n' it is shown unless an     */
/*                       ancestor node had silence set to */
/*                       'y'.                             */
/*                                                        */
/* default               Default text used in the absence */
/*                       of needed variable information.  */
/*                                                        */
/* pro_text              Like 'Pro and Con' this is the   */
/*                       positive answer text template.   */
/*                                                        */
/* con_text              Like 'Pro and Con' this is the   */
/*                       negative answer text template.   */
/*-------------------------------------------------------*/

/*---------------------------------------*/
/* explanation frame for thesis in field rule */
/*---------------------------------------*/
explan_frame(
        id(doa thesis in Field),
        ako(concept1),
        user_level(2),
        silence(n),
        default_text('do a thesis in your field'),
        pro_text('do a thesis in' Field),
        con_text('do not do a thesis in' Field)
```

```
        ).

/*-----------------------------------------*/
/* explanation frame for 'doa thesis' rule    */
/*-----------------------------------------*/
explan_frame(
        id(doa thesis),
        ako(concept2),
        user_level(1),
        silence(n),
        default_text('do a thesis'),
        pro_text('do a thesis'),
        con_text('do not do a thesis')
        ).

/*-----------------------------------------*/
/* explanation frame for 'doa project' rule   */
/*-----------------------------------------*/
explan_frame(
        id(doa project),
        ako(concept3),
        user_level(1),
        silence(n),
        default_text('do a project'),
        pro_text('do a project'),
        con_text('do not do a project')
        ).

/*-------------------------------------------------*/
/* explanation frame for 'doa project in field' rule   */
/*-------------------------------------------------*/
explan_frame(
        id(doa project in Field),
        ako(concept4),
        user_level(2),
        silence(n),
        default_text('do a project in your field of specialty'),
        pro_text('do a project in' Field),
        con_text('do not do a project in' Field)
        ).

/*-----------------------------------------*/
/* explanation frame for 'core done' rule   */
/*-----------------------------------------*/
explan_frame(
        id('you finished the core'),
        ako(concept5),
        user_level(3),
        silence(n),
        default_text('the core courses were completed'),
        pro_text('the core courses were completed'),
        con_text('the core courses were not completed')
        ).

/*-----------------------------------------*/
```

```
/* explanation frame for 'concentration' rule */
/*----------------------------------------*/
explan_frame(
        id(Level 'concentration is' Field),
        ako(concept6),
        user_level(3),
        silence(n),
        default_text('your major/minor concentration is not known'),
        pro_text(your Level 'concentration is' Field),
        con_text(your Level 'concentration is not' Field)
        ).

/*----------------------------------------*/
/* explanation frame for 'seminar hours' rule */
/*----------------------------------------*/
explan_frame(
        id(Field 'seminars totaled' N hours),
        ako(concept7),
        user_level(7),
        silence(n),
        default_text('you have a number of seminar hours in some field'),
        pro_text('you have' N 'seminar hours' in Field),
        con_text('you do not have' N 'seminar hours' in Field)
        ).

/*-------------------------------------*/
/* explanation frame for 'c-level' rule */
/*-------------------------------------*/
explan_frame(
        id(major 'concentration needs' N hours),
        ako(concept8),
        user_level(10),
        silence(y),
        default_text('a major concentration needs at least 4 hours'),
        pro_text('a major concentration needs at least 4 hours'),
        con_text('a major concentration needs at least 4 hours')
        ).

/*-------------------------------------*/
/* explanation frame for 'c-level' rule */
/*-------------------------------------*/
explan_frame(
        id(minor 'concentration needs' N hours),
        ako(concept9),
        user_level(10),
        silence(y),
        default_text('a minor concentration needs at least 2 hours'),
        pro_text('a minor concentration needs at least 2 hours'),
        con_text('a minor concentration needs at least 2 hours')
        ).

/*---------------------------------------------*/
/* explanation frame for 'specialty is' questions */
/*---------------------------------------------*/
explan_frame(
```

```
        id('your specialty is'Field),
        ako(concept10),
        user_level(2),
        silence(n),
        default_text('your specialty is in some field'),
        pro_text('your specialty is' Field),
        con_text('your specialty is not' Field)
        ).

/*-----------------------------------------------*/
/* explanation frame for 'Have N Hours' Fact     */
/*-----------------------------------------------*/
explan_frame(
        id('you have' N hours),
        ako(concept11),
        user_level(2),
        silence(n),
        default_text('you have some hours towards your requirements'),
        pro_text('you have' N 'hours of courses'),
        con_text('you have' N 'hours of courses')
        ).

/*-----------------------------------------------*/
/* explanation frame for 'Core Area' Fact        */
/*-----------------------------------------------*/
explan_frame(
        id('core area' Field),
        ako(concept12),
        user_level(9),
        silence(n),
        default_text('you have taken course in a core area'),
        pro_text('you have a core in' Field),
        con_text(Field 'was not in your core')
        ).

/*-----------------------------------------------*/
/* explanation frame for 'you took Course Fact   */
/*-----------------------------------------------*/
explan_frame(
        id('you took' Course),
        ako(concept13),
        user_level(10),
        silence(n),
        default_text('you took a course or courses'),
        pro_text('you took' Course),
        con_text('you did not take' Course)
        ).

/*-----------------------------------------------*/
/* explanation frame for 'second course' fact    */
/*-----------------------------------------------*/
explan_frame(
        id(ai 'second course is' 782),
        ako(concept15),
        user_level(10),
```

```
        silence(y),
        default_text('the second course for the ai core is 782'),
        pro_text('the second course for the ai core is 782'),
        con_text('the second course for the ai core is 782')
        ).

/*-----------------------------------------------*/
/* explanation frame for 'second course' fact    */
/*-----------------------------------------------*/
explan_frame(
        id(os 'second course is' 810),
        ako(concept15),
        user_level(10),
        silence(y),
        default_text('the second course for the os core is 810'),
        pro_text('the second course for the os core is 810'),
        con_text('the second course for the os core is 810')
        ).

/*-----------------------------------------------*/
/* explanation frame for 'second course' fact    */
/*-----------------------------------------------*/
explan_frame(
        id(theory 'second course is' 850),
        ako(concept15),
        user_level(10),
        silence(y),
        default_text('the second course for the theory core is 850'),
        pro_text('the second course for the theory core is 850'),
        con_text('the second course for the theory core is 850')
        ).

/*-----------------------------------------------*/
/* explanation frame for 'second course' fact    */
/*-----------------------------------------------*/
explan_frame(
        id(languages 'second course is' 711),
        ako(concept15),
        user_level(10),
        silence(y),
        default_text('the second course for the languages core is 711'),
        pro_text('the second course for the languages core is 711'),
        con_text('the second course for the languages core is 711')
        ).

/*-----------------------------------------------*/
/* explanation frame for 'second course' fact    */
/*-----------------------------------------------*/
explan_frame(
        id(networking 'second course is' 745),
        ako(concept15),
        user_level(10),
        silence(y),
        default_text('the second course for the networking core is 745'),
        pro_text('the second course for the networking core is 745'),
```

```
        con_text('the second course for the networking core is 745')
        ).

/*-------------------------------------------*/
 /* explanation frame for 'second course' fact    */
/*-------------------------------------------*/
 explan_frame(
        id(hardware 'second course is' 721),
        ako(concept15),
        user_level(10),
        silence(y),
        default_text('the second course for the hardware core is 721'),
        pro_text('the second course for the hardware core is 721'),
        con_text('the second course for the hardware core is 721')
        ).
```

```
/*-------------------------------------------------------*/
/* get_explan(Id,Truth,Explan_structure)                 */
/*                                                       */
/*  Given Explan_frame return an explanation structure to be*/
/*  used in the actual answer presented to the user.  This */
/*  involves obtaining the defaults and assigning them to  */
/*  the appropriate variables.  The explan_structure is as */
/*  follows:                                             */
/*                                                       */
/*  explan_structure(Text,Level,Silence,Concept_id)      */
/*                                                       */
/*  where:                                               */
/*                                                       */
/*    Text      text to be printed to user.              */
/*    Level     user level that this text applies to.    */
/*    Silence   y or n (yes or no) to determine if shown. */
/*    Concept_id    general concept for summarization.   */
/*                                                       */
/*  Text can be pro, con, or default.  Choice depends on  */
/*  the information provided by the user and the existance */
/*  of a flag ( pro_or_con(pro) or pro_or_con(con) ).     */
/*-------------------------------------------------------*/

 get_explan(Id,Truth,Explan_structure) :-

     obtain_frame(Id,Explan_frame),
     Explan_frame = explan_frame(id(Id),
                                 ako(Concept_id),
                                 user_level(Level),
                                 silence(Silence),
                                 default_text(Default_text),
                                 pro_text(Pro_text),
                                 con_text(Con_text)),

     (
       Truth == true,
       $_format_goal(Pro_text,Filled_text),   /* replace uninstantiated */
                                              /* parts with defaults.   */
       Final_text = Filled_text
      ;
       Truth == false,
       $_format_goal(Con_text,Filled_text),  /* replace uninstantiated */
                                             /* parts with defaults.   */
       Final_text = Filled_text
     ),
     /*---------------------------------------*/
     /* set up return structure now            */
     /*---------------------------------------*/

      Explan_structure = explan_structure(Final_text,Level,Silence,
                                          Concept_id, Truth).
```

```
  /*-------------------------------------------------------------*/
  /* if the explanation is all silent or suppressed by the threshold*/
  /* or a combination of both the user would not see anything.  Let */
  /* the user know what is going on in this case and try a different*/
  /* threshold to see if this solves the problem.                */
  /*-------------------------------------------------------------*/
stuff_to_show(Solution, Threshold) :-

   $_no_more_exp(Solution,Threshold),
   ( Threshold < 10,
     write('  The explanation for this solution cannot be shown because '),nl,
     write('the information is suppressed.   Please try a higher value '),nl,
     write('detail level than '),write(Threshold), write(' since this may '),
     nl,
     write('permit some explanation to be presented. '), nl,nl,
    !,
    fail
    ;
     write('All explanation is silenced. Contact the Knowledge Engineer '),
     nl,
     write('to correct this problem. '),
     !,
     fail
   )
   ;
    !.
```

```
/*-----------------------------------------------------------------*/
/* $_no_more_exp  No More Explanation check.   If all remaining    */
/* explanation structures are silent or below the threshold for the user */
/* then no more explanation is available and the clauses succeed.  */
/*-----------------------------------------------------------------*/

$_no_more_exp( Answer1 and Answer2, Threshold) :-
  !,
  $_no_more_exp(Answer1,Threshold),
  $_no_more_exp(Answer2,Threshold).

$_no_more_exp( Answer1 but Answer2, Threshold) :-
  !,
  $_no_more_exp(Answer1,Threshold),
  $_no_more_exp(Answer2,Threshold).


/*-----------------------------------------------------------*/
/* silence has precedence over the user level.  If node is   */
/* silent then the sub-tree is as well and need not be examined */
/*-----------------------------------------------------------*/
$_no_more_exp( Answer was Found , Threshold) :-
  Answer = (explan_structure(_,User_level,Silence,_,_)  ),
  Silence = y.

/*-----------------------------------------------------------*/
/* test threshold against user level.   If the user level makes */
/* this portion of the explanation silent the subtree may or may*/
/* not be silent so examine the subtree as well.             */
/*-----------------------------------------------------------*/
$_no_more_exp( Answer was Found , Threshold) :-
  Answer = (explan_structure(_,User_level,Silence,_,_)  ),
  Silence = n,
  User_level > Threshold,
  (
    Found = Derived from Answer2,          /* this is a rule so a */
    $_no_more_exp(Answer2,Threshold)        /* sub-tree exists.    */
   ;
    true                                   /* no subtree, succeed */
  ).
```

```
/*-------------------------------------------------------*/
/* obtain_frame(Id,Explan_frame)                         */
/*                                                       */
/*  Given 'Id' match and explan_frame in the data base   */
/*  and return the structure to the invoking routine.    */
/*-------------------------------------------------------*/

  obtain_frame(Id,Explan_frame) :-
            explan_frame(id(Id),ako(Concept),user_level(Level),
                         silence(Silence),
                         default_text(Default),pro_text(Pro_text),
                         con_text(Con_text)),

            Explan_frame = explan_frame(id(Id),
                                        ako(Concept),
                                        user_level(Level),
                                        silence(Silence),
                                        default_text(Default),
                                        pro_text(Pro_text),
                                        con_text(Con_text)).
```

```
/*-------------------------------------------------------------*/
/* present_explan(Answer)                                      */
/*                                                             */
/*    Answer     List containing explan_structures to be       */
/*               presented as an explanation to the user.      */
/*- -    -    - -      -        -          - -   -      -*/
/* Operation:                                                  */
/*    The bulk of the explanation facility is involved here.   */
/* Presenting the answer is not a simple matter of printing text.*/
/* Initially the user is prompted for the level of detail desired*/
/* which is in the range of 1 through 10 or an 's' for summary. */
/* Three actions must be checked prior to printing text,they are */
/* exception cases that need addressing.  First is an all       */
/* silent explanation.   If all clauses retrieved are marked as */
/* silent then the user would not see anything and be mystified! */
/* In this case the system prompts the user asking if an override*/
/* is desired which will result in a full explanation down to the*/
/* gritty details.   The second check if for a similar condition */
/* caused by choosing a user level that is below all items in the*/
/* explanation, again causing nothing to be printed.   In this  */
/* case the user is instructed to chose a number between the    */
/* lowest value that is in the list (user_level) and 10 so that */
/* something is shown.   The last case deals with the user_level */
/* as well.   If the user desires they can select a level of 's' */
/* which indicates a desire to see a summary.   A summary is    */
/* presented by obtaining the AKO concepts and printing them.   */
/*                                                             */
/*     The general order of processing once the exception checks */
/* are handled is to check if the particular item is silent. If */
/* it is then go on to the next item of text, if not, the level */
/* selected by the user must be compared to the user_level in the*/
/* structure.   If the user_level in the structure is less than */
/* or equal to the one selected by the user then the item is    */
/* displayed.   At this point it is now know whether or not the */
/* current structure has text to be displayed.   After the text */
/* is printed the 'linking' phrase (see the phrases above) must */
/* be determined.   This occurs by examining the connecting oper-*/
/* ator in Answer.  'And' indicates support and 'but' indicates */
/* contradictory information.   Rules are recognized by the     */
/* existance of the operator 'from' and wording is appropriate  */
/* for the rule level.                                         */
/*                                                             */
/* The process continues until the list of structures is       */
/* exhausted.                                                  */
/*-------------------------------------------------------------*/

present_explan(Answer) :-
      nl,
      $_showconclusion(Answer),              /* Give the answer  */
      nl,
      write('Would you like to see how? '),  /* Ask user for how */
      getreply( ynw, Reply), !,
      ( Reply == yes,
        $_show(Answer),
```

```
        nl,
        !
      ;
        true
      ).

$_showconclusion( Answer1 was Found) :-  /* Rule leads answer */
    !,
   (
    top_goal(Top_goal_statement)
   ;
    Top_goal_statement = 'determine whether to '   /* in case top goal absent */
  ),
    nl,
    write('The intent was to '),
    write(Top_goal_statement),
    nl,
    write('The answer is '),
    Answer1 = explan_structure(Text,_,_,_,_),
    write(Text),
    nl.

$_showconclusion( Answer1 and Answer2) :-
   !,
   $_showconclusion(Answer1),
   write(' and '),
   $_showconclusion(Answer2).

$_showconclusion( Answer1 but Answer2) :-
   !,
   $_showconclusion(Answer1),
   write(' but '),
   $_showconclusion( Answer2).
```

```
$_show(Solution) :-
  nl,
  write('Enter an "s" if you would like to see a summary, any other letter'),
  nl,
  write('to see a full explanation.    Enter choice: '),
  read(Summary_or_full),
  repeat,
  nl,
  nl,
  write('Enter a number from 1 to 10 to select the amount of detail you'),nl,
  write('want to see, 1 being the least detailed.  Enter level of detail: '),
  read(Threshold),
  nl,
  stuff_to_show(Solution,Threshold),   /* warn user if there is nothing */
                                       /* to show.                      */
/*-----------------------------------*/
/* choose summary or full explanation */
/*-----------------------------------*/
  (  Summary_or_full == s,
     summarize(Solution,Threshold)
   ;
     reset_line_length(0),             /* for use in new_line */
     $_show(Solution, Threshold, 0)
  ),
  !.


/*-------------------------------------------------------------------*/
/* Handle explanation of conjunctions.                               */
/*-------------------------------------------------------------------*/

$_show( Answer1 and Answer2, Threshold, Nesting_level) :-

   !,
   Answer1 = (explan_structure(_,User_level,Silence,Concept_id,_) was _ ),
     (
       Silence   n,                    /* Not silent          */
       User_level =< Threshold,        /* and withing limits  */
       $_show(Answer1, Threshold, Nesting_level)
      ;
       Answer1_is_silent = y,          /* nothing to show here */
       !
    ),

   /*-----------------------------------------------*/
   /*  and more explanation to do for the other part ? */
   /*-----------------------------------------------*/

     (
       $_no_more_exp( Answer2, Threshold),
       !
```

```
    ;
     ( Answer1_is_silent == y,
       !
     ;
       new_line(' and '),
       write(' and ')
     ),
     $_show( Answer2, Threshold, Nesting_level)
    ).

/*-----------------------------------------------------------------*/
/* Handle explanation of disjunctions.                             */
/*-----------------------------------------------------------------*/

$_show( Answer1 but Answer2, Threshold, Nesting_level) :-

   !,
   Answer1 = (explan_structure(_,User_level,Silence,Concept_id,_) was _ ),
    (
       Silence = n,                      /* Not silent          */
       User_level =< Threshold,          /* and withing limits  */
       $_show(Answer1, Threshold, Nesting_level)
     ;
       Answer1_is_silent - y,            /* nothing to show here */
       !
    ),

   /*-------------------------------------------------*/
   /*  and more explanation to do for the other part ? */
   /*-------------------------------------------------*/

     (
       $_no_more_exp( Answer2, Threshold),
       !
     ;
      ( Answer1_is_silent == y,
        !
      ;
        new_line(' but '),
        write(' but ')
      ),
     $_show( Answer2, Threshold, Nesting_level)
    ).

/*-----------------------------------------------------------------*/
/* non-compound case handler. Check first for silent or below threshold  */
/* If silent or below threshold then do nothing, else print text.        */
/*-----------------------------------------------------------------*/
 $_show(Answer was Found ,Threshold, Nesting_level) :-

      Answer = (explan_structure(Text,User_level,Silence,Concept_id,_)),
      (
        Silence = n,
        User_level =< Threshold,
        new_line(Text),
```

```
        write(Text),

        $_show1(Found,Threshold,Nesting_level,Text)
        ;
        true,                              /* nothing to show */
        !
      ).

/*-------------------------------------------------------------------*/
/* Uninstantiated catch all.                                         */
/*-------------------------------------------------------------------*/
 $_show(Answer,Threshold,_) :-
     var(Answer),
     new_line('empty concept'),
     write('empty concept').

/*-------------------------------------------------------------------*/
/* Handle explanation of a rule                                      */
/*-------------------------------------------------------------------*/

$_show1(Derived_by from Answer,Threshold, Nesting_level,Rule_text) :-
     (
       $_no_more_exp( Answer, Threshold)
     ;
       new_line(' which is supported by '),
       write(' which is supported by '),
       Nesting_level1 is Nesting_level + 1,
       (
         !,
         $_show( Answer, Threshold, Nesting_level1),
         (
            Nesting_level1 > 0,
            new_line(' This was the support for '),
            write(' This was the support for '),
            new_line(Rule_text),
            write(Rule_text),
            new_line('.   '),
            write('.   ')
            ;
            true,        /* end of summarization so do not print link */
            !            /* phrase of THIS IS ALL THE SUPPORT FOR     */

         )
       )
     ).

 $_show1( Found,_,_,_) :- !.   /* all other 'stuff' goes to a black*/
                               /* hole.                            */
```

```
/*-------------------------------------------------------------*/
/* summarize(Answer,Threshold)  Summarization requires obtaining  */
/* the text that goes along with the concept referenced in the    */
/* explan_structure.    Also noted is the Silence and user-level  */
/* of the explan_structure.   If the Silence - 'y' or the         */
/* user level is below the threshold the summarization text is not*/
/* included.   For simplicity, if summarization text cannot be    */
/* located it is simply not printed rather than a failure.        */
/* the Concept structure is Concept(Concept_id,Line,Text) where   */
/* Concept_id matches the concept id. in the explan_structure,    */
/* Line is the line number from 1 to whatever you feel like ( or  */
/* the limits of the computers integer range, and Text is the text*/
/* to be printed.                                                 */
/*-------------------------------------------------------------*/
 summarize(Answer,Threshold) :-
      $_summary(Answer, Threshold, 0).

/*-------------------------------------------------------------------*/
/* Handle summary of rule.                                           */
/*-------------------------------------------------------------------*/

$_summary1(Derived_by from Answer,Threshold, Nesting_level,Rule_text) :-
     (
       $_no_more_exp( Answer, Threshold)
     ;
       new_line(' This is supported by '),
       write(' This is supported by '),
       Nesting_level1 is Nesting_level + 1,
       (
         !,
         $_summary( Answer, Threshold, Nesting_level1),
         (
            Nesting_level1 > 0,
            new_line(' This is all the support for '),
            write(' This is all the support for '),
            new_line(Rule_text),    /* passed during invokation */
            write(Rule_text)
            ;
            true,        /* end of summarization so do not print link */
            !            /* phrase of IS ALSO SUPPORTED BY            */

         )
       )
     ).

 $_summary1( Found, _,_,_) :- !.   /* all other 'stuff' goes to a black*/
                                /* hole.                             */

/*-----------------------------------------------------------------*/
/* Handle explanation of conjunctions.                             */
/*-----------------------------------------------------------------*/

$_summary( Answer1 and Answer2, Threshold, Nesting_level) :-
```

```
   !,
   Answer1 = (explan_structure(_,User_level,Silence,Concept_id,_) was _),
    (
       Silence - n,                        /* Not silent          */
       User_level =< Threshold,            /* and withing limits  */
       $_summary(Answer1, Threshold, Nesting_level)
      ;
       Answer1_silent = y,
       !
    ),

   /*-------------------------------------------------*/
   /*  and more explanation to do for the other part ? */
   /*-------------------------------------------------*/
    (
       $_no_more_exp( Answer2, Threshold),
       !
     ;
       $_summary( Answer2, Threshold, Nesting_level)
   ).

/*-------------------------------------------------------------------*/
/* Handle explanation of disjunctions.                               */
/*-------------------------------------------------------------------*/

$_summary( Answer1 but Answer2, Threshold, Nesting_level) :-

   !,
   Answer1 = (explan_structure(_,User_level,Silence,Concept_id,_) was _),
    (
       Silence = n,                        /* Not silent          */
       User_level =< Threshold,            /* and withing limits  */
       $_summary(Answer1, Threshold, Nesting_level)
      ;
       true,                               /* nothing to show here */
       !
    ),

   /*-------------------------------------------------*/
   /*  and more explanation to do for the other part ? */
   /*-------------------------------------------------*/
    (
       $_no_more_exp( Answer2, Threshold),
       !
     ;
       $_summary( Answer2, Threshold, Nesting_level)
   ).

/*-------------------------------------------------------------------*/
/* non-compound case handler. Check first for silent or below threshold */
/* If silent or below threshold then do nothing, else print text.    */
/*-------------------------------------------------------------------*/
 $_summary(Answer was Found ,Threshold, Nesting_level) :-

      Answer = (explan_structure(Text,User_level,Silence,Concept_id,_)),
```

```
     (
       Silence - n,
       User_level =< Threshold,
     (
       not concept_text(Concept_id,_,_),
       !,
       new_line(' '),
       write(' '),
       new_line(Text),
       write(Text)      /* if you can't find the concept at least let them */
                        /* see something.   This is the norm for primative */
                        /* items such a fact since the concept is self-    */
                        /* explanatory.                                    */
     ;
       write_concept(Concept_id)
     ),
       $_summary1(Found,Threshold,Nesting_level,Text)
       ;
       true,                            /* nothing to show */
       !
     ).

/*-------------------------------------------------------------------------*/
/* Uninstantiated catch all.                                               */
/*-------------------------------------------------------------------------*/
 $_summary(Answer,Threshold, _) :-
     var(Answer),
     new_line('empty concept'),
     write('empty concept').

/*---------------------------------------------------------*/
/* write_concept.   While there are lines available, print  */
/* the concept.  THIS ASSUMES THE KNOWLEDGE ENGINEER DID NOT*/
/* skip line numbers when entering text.   A routine should */
/* be created to insure this!                               */
/*---------------------------------------------------------*/
 write_concept(Concept_id) :-
     write_concept(Concept_id,1).    /* from the 1st line */

 write_concept(Concept_id,Line) :-
     not concept_text(Concept_id,Line,_).

 write_concept(Concept_id,Line) :-
     concept_text(Concept_id,Line,Text),
     new_line(Text),
     write(Text),
     Next_line is Line + 1,
     write_concept(Concept_id,Next_line).
```

Appendix 3
Turbo Prolog Expert System Shell

```
/*-----------------------------------------*/
/* first phase in defining an expert system */
/* shell in turbo prolog.   Primary intent  */
/* is to trace the operations for better    */
/* understanding.                           */
/*-----------------------------------------*/
/*trace*/
domains
  list = symbol*

database
  info(symbol,list)
  yes(symbol)
  no(symbol)

predicates
  append(list,list,list)
  writelist(list,integer)
  purge
  add(symbol,list,list)
  query
  attributes(symbol,list)
  process(symbol,symbol,symbol)
  start
  trailyes(list)
  trailno(list)
  xtrailyes(symbol,list,list)
  xtrailno(symbol,list,list)
  try(symbol,list)
  enter
  member(symbol,list)
  xwrite(symbol)

goal
  start.

clauses

  start :-
    write("Knowledge base to consult please: "),
    readln(KnowledgeBase),
    consult(KnowledgeBase),
    fail.
```

```
start :-
  assert(yes(end)),
  assert(no(end)),
  write("Enter Knowledge? (y/n): "),
  readln(A),
  A=y,
  not(enter),
  write("Name of Knowledge base to save to: ( xxx.dat ) "),
  readln(KnowledgeBase),
  save(KnowledgeBase),!,query.
start :-
  query,
  purge.
start :-
  purge.

enter :-
  write("What is the object? "),
  readln(Object),
  Object <> "quit",
  attributes(Object,[]),
  write("     more? (y/n) "),!,
  readln(Q),Q="y",
  enter.

attributes(O,List) :-
  write(O,"'s next attribute please:(or quit) "),
  readln(Attribute),
  Attribute <> "quit",
  add(Attribute,List,List2),
  attributes(O,List2).

attributes(Object,List) :-
  assert(info(Object,List)),
  writelist(List,1),!,
  nl.

add(X,L,[X|L]).

query :-
  info(O,A),
  trailyes(A),
  trailno(A),
  try(O,A),
  purge.

query :- purge.
```

```
trailyes(A) :-
  yes(T),
  !,
  xtrailyes(T,A,[]),
  !.

xtrailyes(end,_,_) :- !.
xtrailyes(T,A,L) :-
  member(T,A), !,
  add(T,L,L2),
  yes(X),not(member(X,L2)),
  xtrailyes(X,A,L2).

trailno(A) :-
  no(T),!,
  xtrailno(T,A,[]),!.

xtrailno(end,_,_) :- !.
xtrailno(T,A,L) :-
  not(member(T,A)),!,
  add(T,L,L2),
  no(X),
  not(member(X,L2)),!,
  xtrailno(X,A,L2).

try(O,[]) :- write(" It is a(n) ",O), nl.
try(O,[X|T]) :-
  yes(X),!,
  try(O,T).
try(O,[X|T]) :-
  write("is ",X," an attribute of the object. (y/n/why) ?"),
  readln(Q),
  process(O,X,Q),!,
  try(O,T).

process(_,X,y) :-
  asserta(yes(X)),!.
process(_,X,n) :-
  asserta(no(X)),!,fail.
process(O,X,why) :-
  write("I think it may be"),nl,
  write(O," because it has: "),nl,
  yes(Z),xwrite(Z),nl,
  Z=end,!,
  write("is ",X," an attribute of ",O," (y/n/why) ?"),
  readln(Q),
  process(O,X,Q),!.
```

```
xwrite(end).
xwrite(X) :-
  write(X).

purge :-
  retract(yes(X)),
  X=end,
  fail.
purge :-
  retract(no(X)),
  X=end.

append([],List,List).
append([X|L1], List2, [X|L3]) :-
  append(L1,List2,L3).

writelist([],_).
writelist([Head|Tail],3) :-
 write(Head),nl,writelist(Tail,1).
writelist([Head|Tail],I) :-
 N = I+1,
 write(Head," "),writelist(Tail,N).

member(N,[N|_]).
member(N,[_|T]) :- member(N,T).
```

Appendix 4

Example Explanations from the Original and Modified B-Shell